Hashing Extensível e Linear

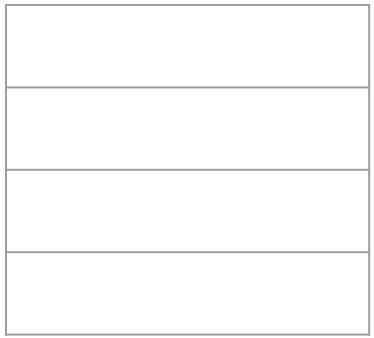
Prof. José J. Camata
Prof. Marcelo Caniato
camata@ice.ufjf.br
marcelo.caniato@ice.ufjf.br

Introdução

- Rehashing permite redimensionar uma tabela hash
  - No entanto, o custo pode ser muito alto para algumas aplicações
- E se pudéssemos fazer apenas um *rehashing* local?
  - Não possível com vetores, mas possível com arquivos!
  - Duas classes de técnicas
    - Baseadas em diretório
      - Hashing expansível (Knott, 1971)
      - Hashing dinâmico (Larson, 1978)
      - **Hashing extensível (Fagin et al., 1979)**
    - Sem diretório
      - Hashing virtual (Litwin, 1978)
      - **Hashing linear (Litwin, 1978)**

Hashing extensível

- Arquivo dinâmico composto de **baldes**

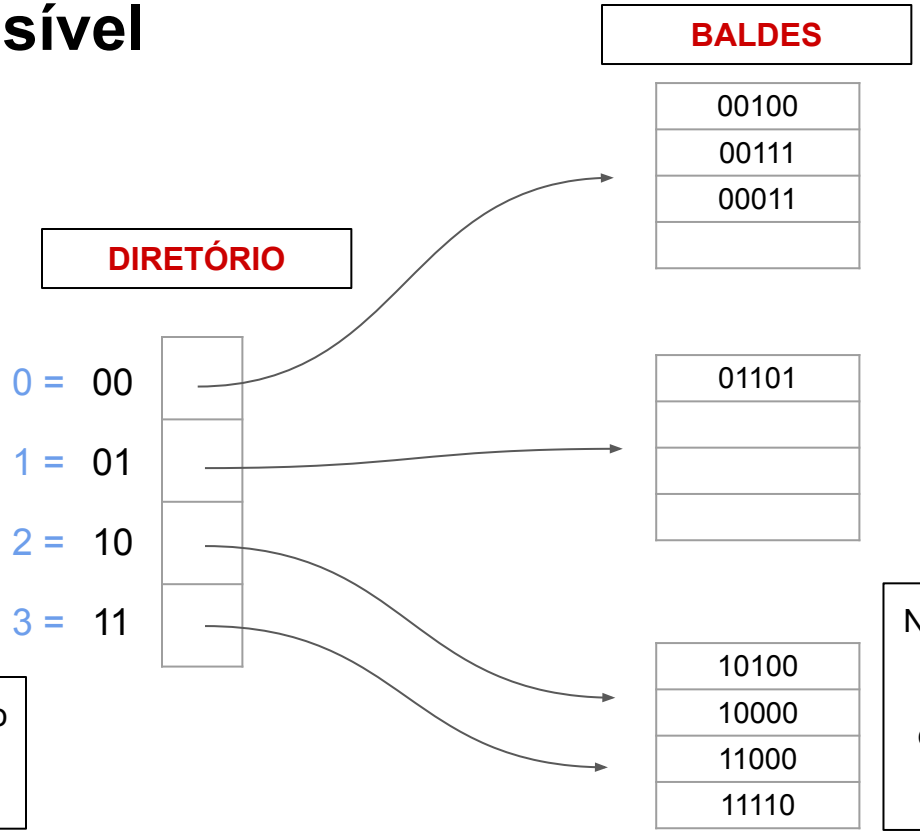


O balde possui um tamanho fixo M, armazenando, portanto, no máximo M chaves

- Dados acessados indiretamente através de um índice
  - h(k) indica uma posição no índice, e não no arquivo
    - Valores retornados por h(k) são chamados de **pseudochaves**

Hashing extensível

- Exemplo

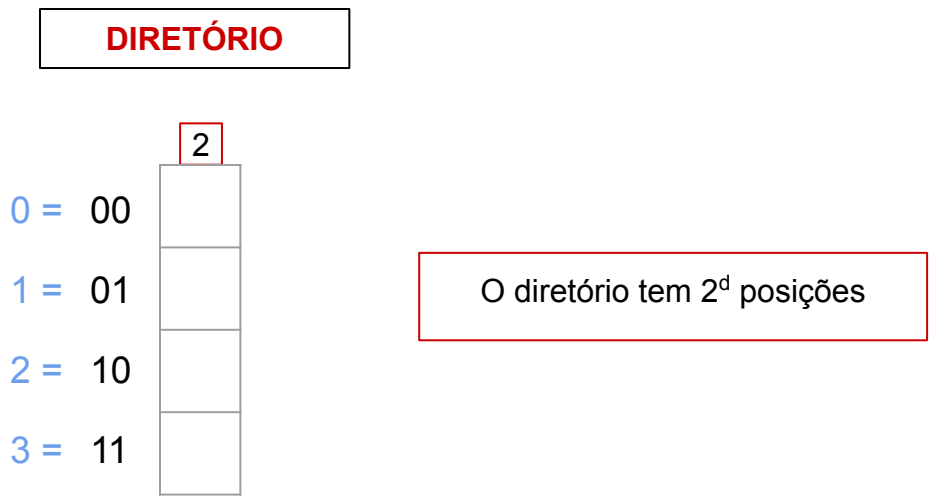


Mais de uma posição do diretório pode apontar para um mesmo balde

Note que todas as chaves de um determinado balde possuem como prefixo os 2 bits de uma das posições do dicionário que apontam para ele

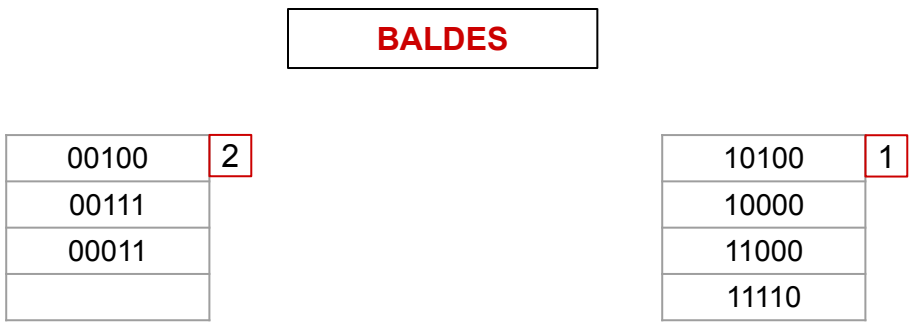
Hashing extensível

- **Profundidade do diretório**: número de bits da *pseudochave* que são usados para identificar um balde
  - Utiliza-se os *d* bits mais à esquerda da *pseudochave*



Hashing extensível

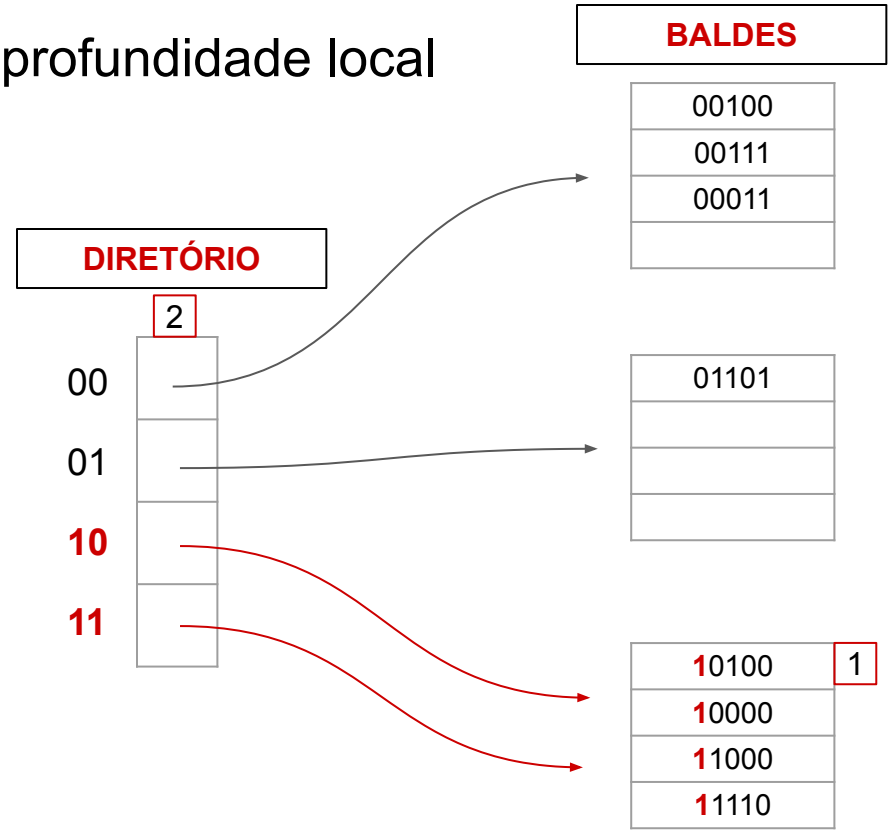
- **Profundidade local**: número de bits que todas as *pseudochaves* de um balde têm em comum
  - Novamente, utiliza-se os *d* bits mais à esquerda da *pseudochave*
  - Se a profundidade local de um balde é *d*, então todas as *pseudochaves* deste balde possuem os mesmos *d* bits iniciais



Hashing extensível

- Profundidade global X profundidade local

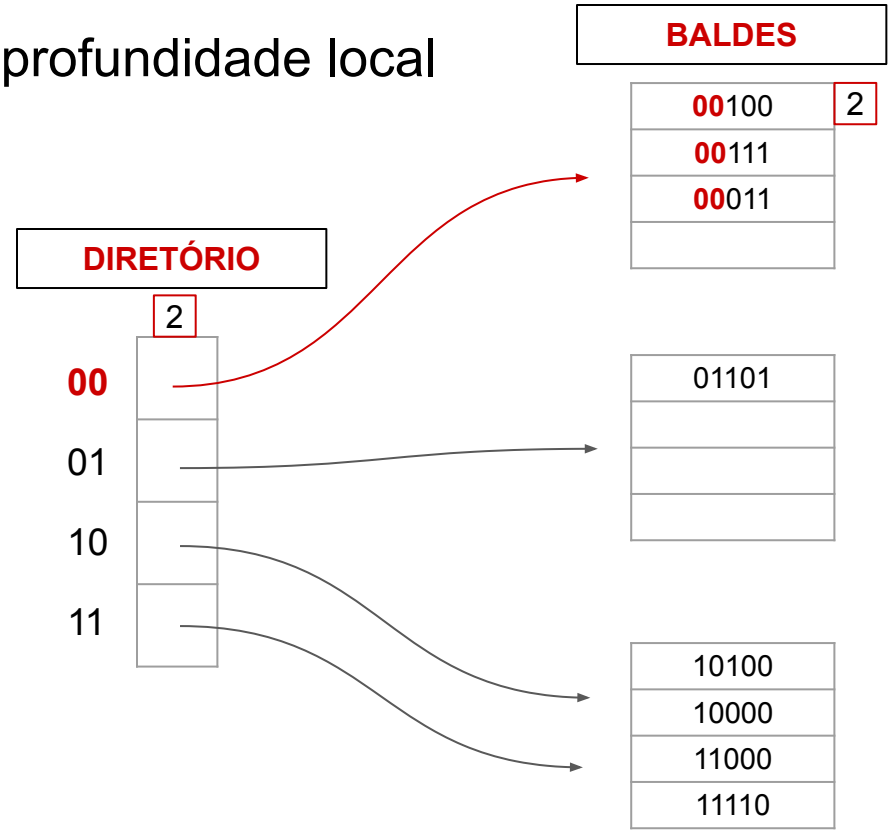
$d_{local} < d_{global}$



Hashing extensível

- Profundidade global X profundidade local

$d_{local} = d_{global}$

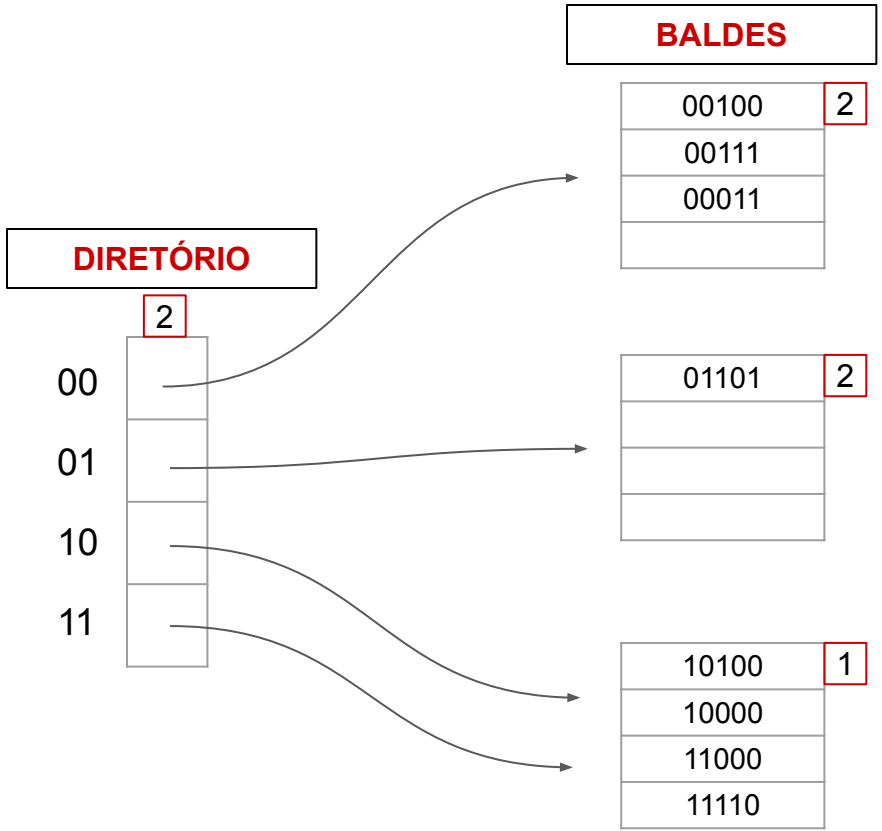


Hashing extensível

- Busca
  1. Aplica a função h(k) sobre a chave k
  2. Obtém a profundidade d do diretório
  3. Obtém os *d* bits mais à esquerda da *pseudochave* h(k)
  4. Acessa o balde associado aos d bits obtidos
  5. Busca a *pseudochave* no balde, retornando verdadeiro caso ela seja encontrada, ou falso em caso contrário

Hashing extensível

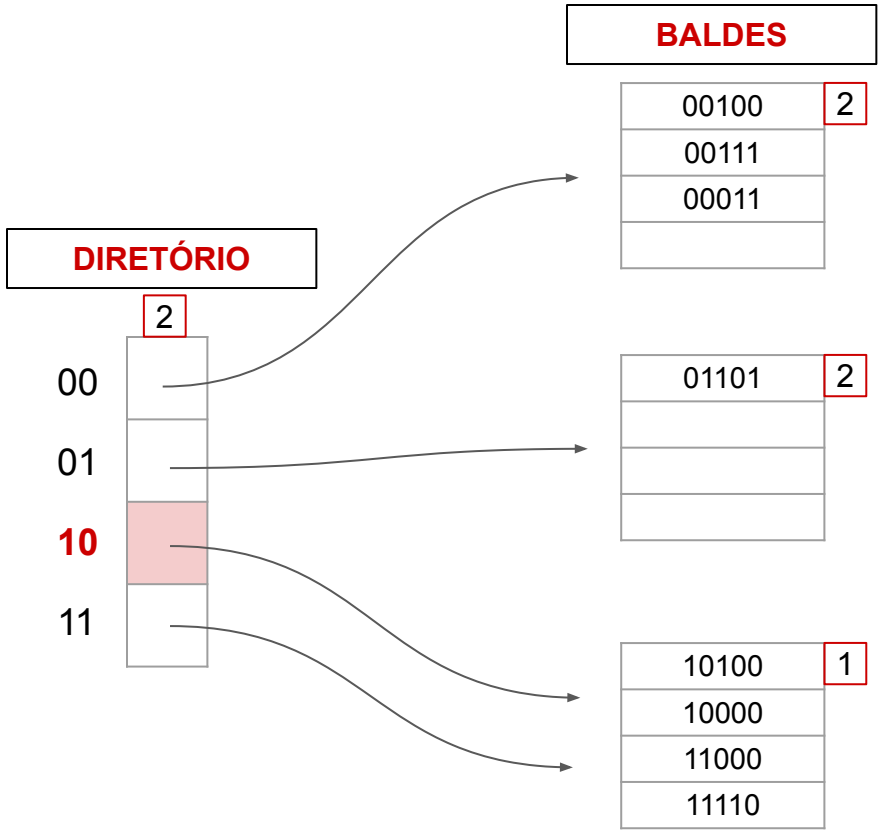
➡ **Buscar 10100**



Hashing extensível

Buscar 10100

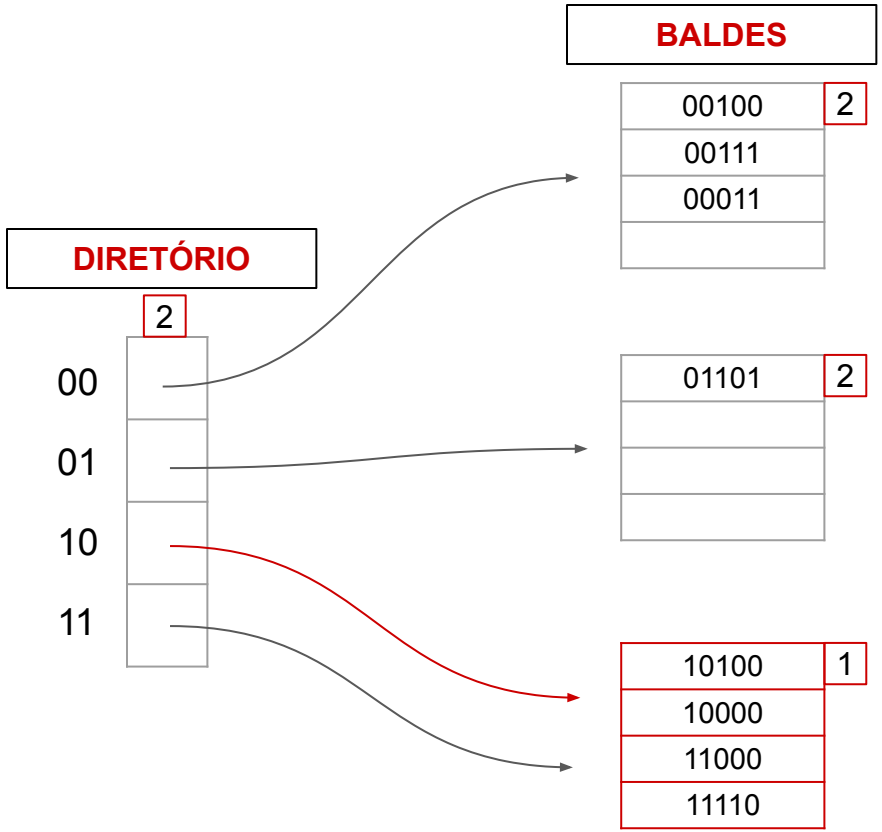
1. Identifica os d bits mais à esquerda da *pseudochave*



Hashing extensível

Buscar 10100

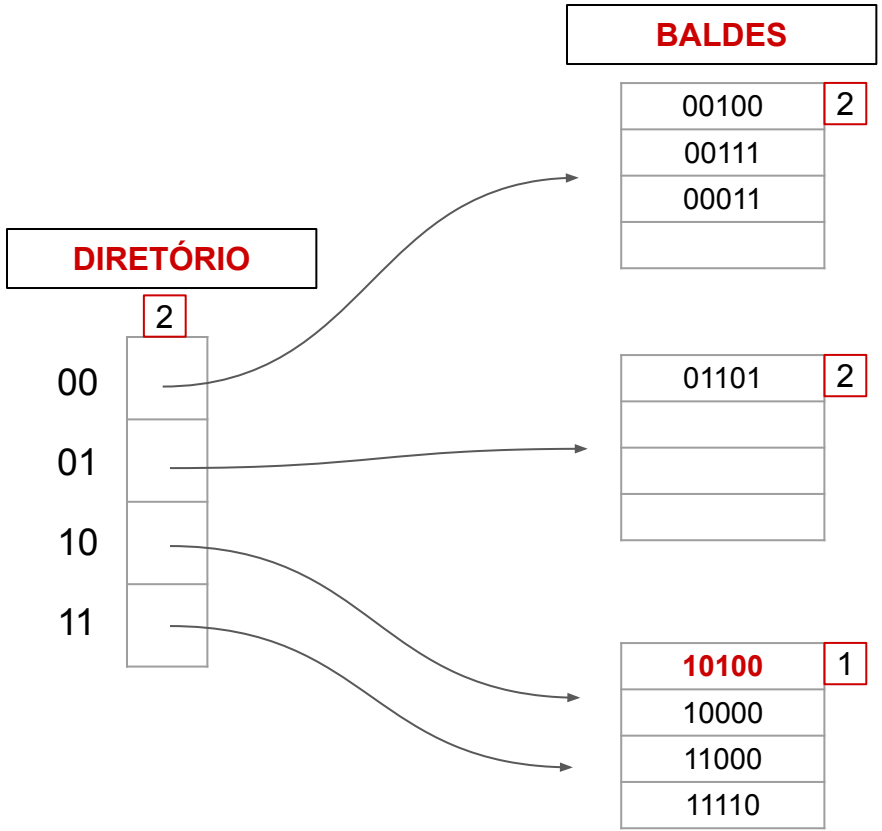
1. Identifica os d bits mais à esquerda da pseudochave
2. **Acessa o balde apontado por essa posição no diretório**



Hashing extensível

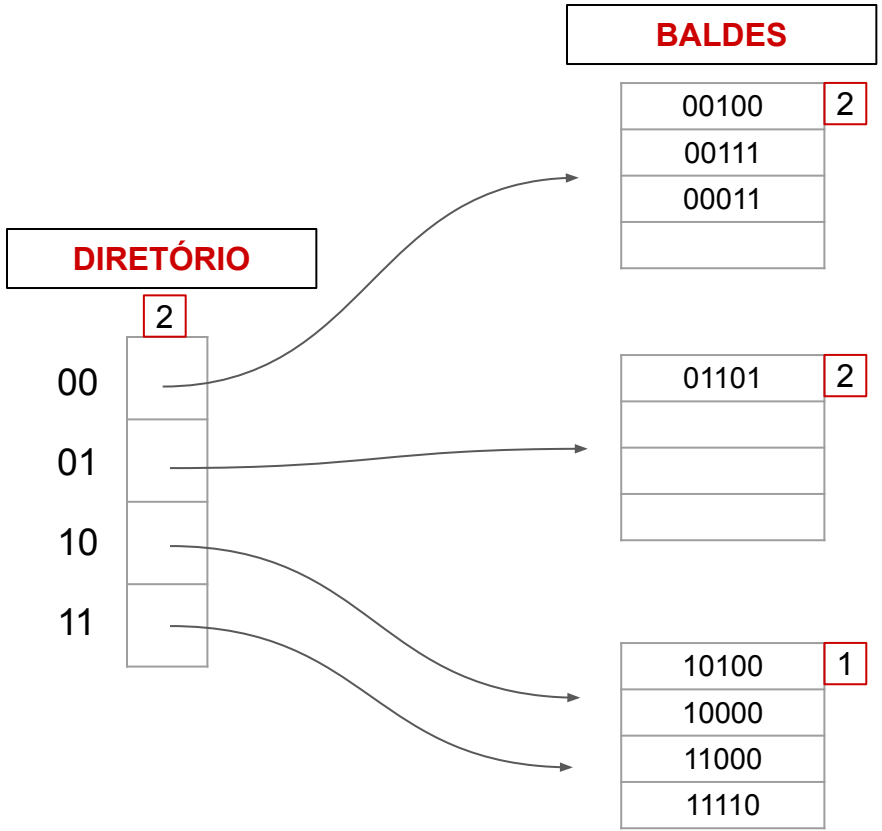
Buscar 10100

1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Busca a chave no balde**



Hashing extensível

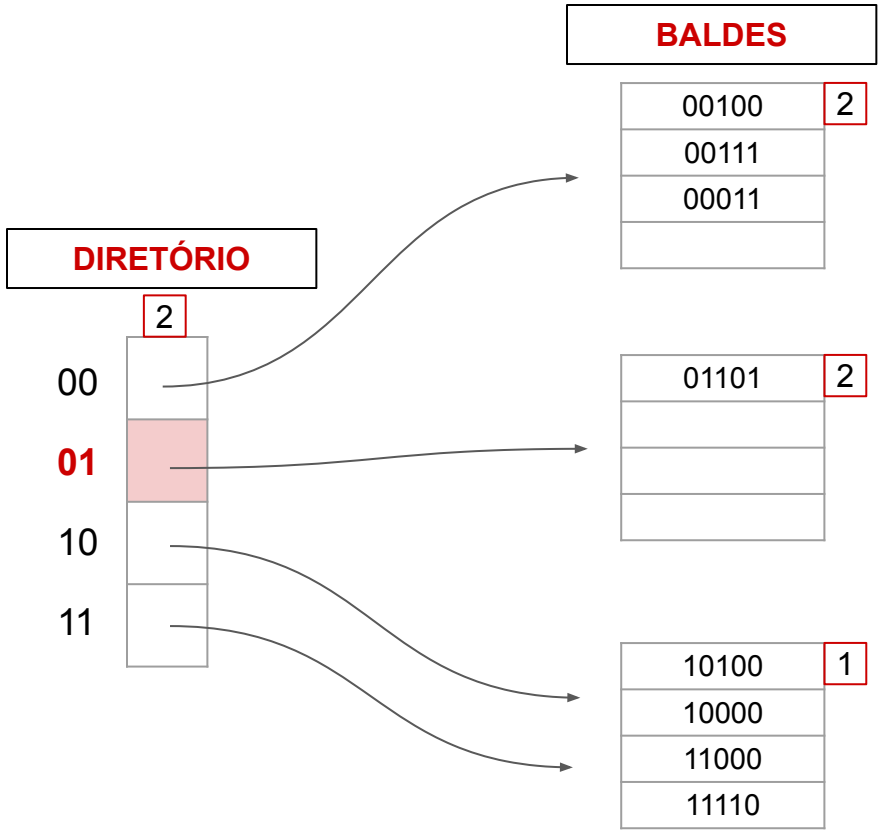
- Buscar 01100



Hashing extensível

Buscar 01100

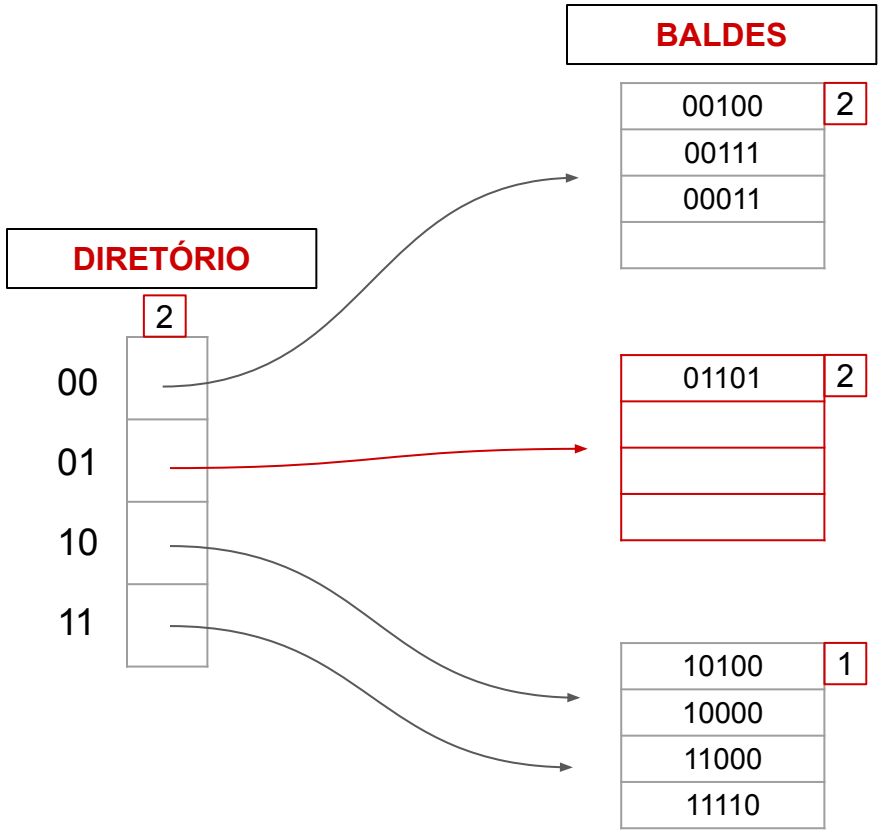
1. Identifica os d bits mais à esquerda da pseudochave



Hashing extensível

Buscar 01100

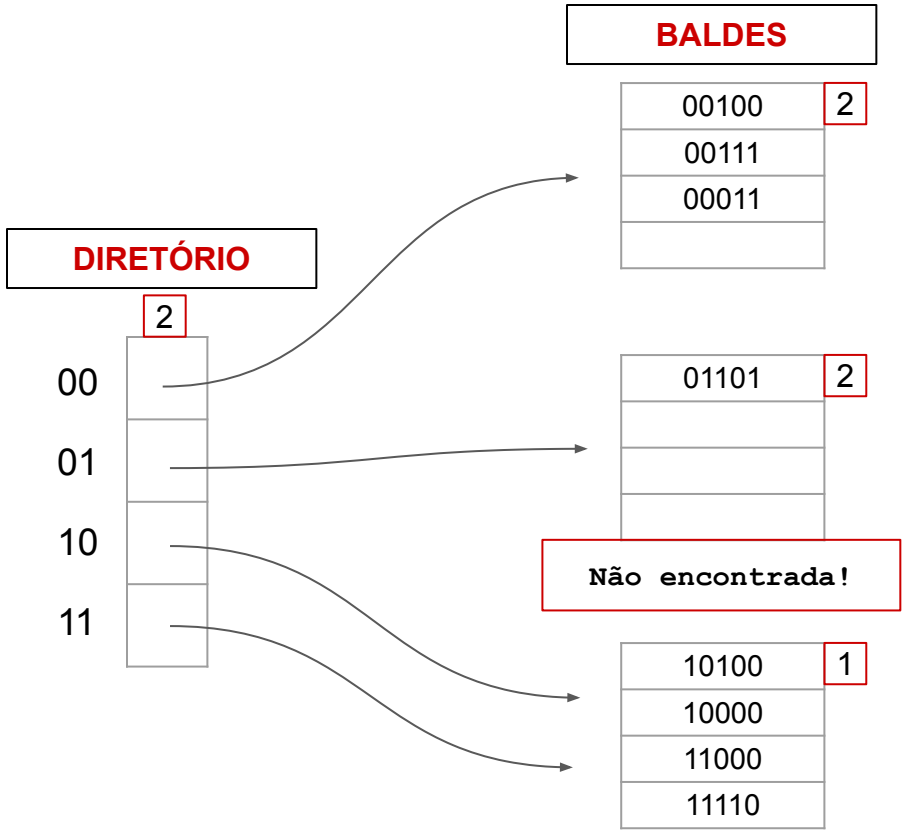
1. Identifica os d bits mais à esquerda da pseudochave
2. **Acessa o balde apontado por essa posição no diretório**



Hashing extensível

Buscar 01100

1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Busca a chave no balde**



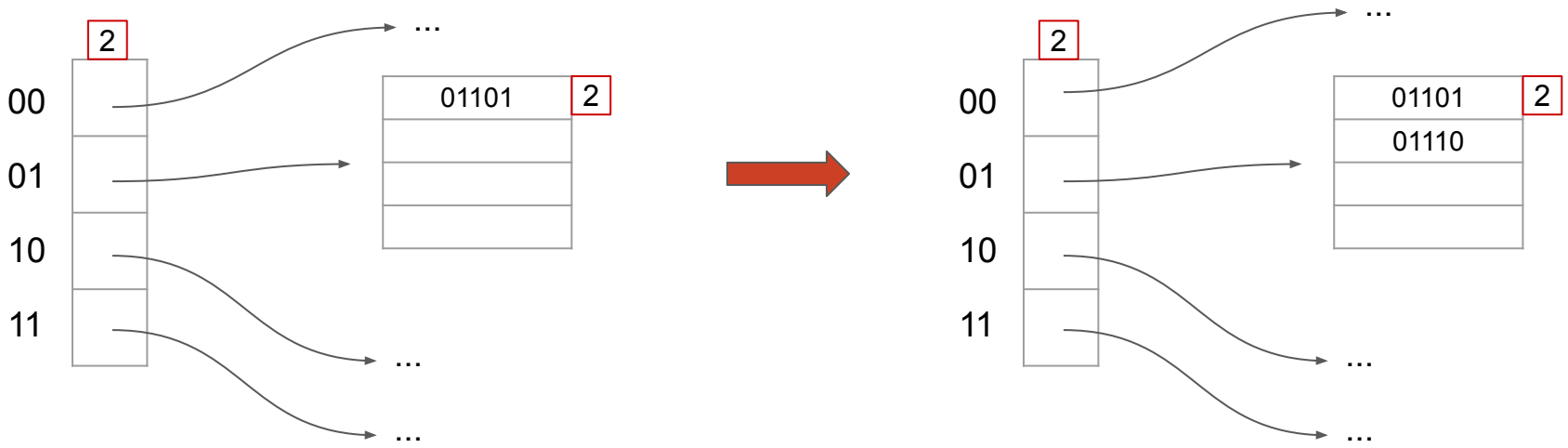
Hashing extensível

- **Inserção**
  1. Aplica a função h(k) sobre a chave k
  2. Obtém a profundidade d do diretório
  3. Obtém os *d* bits mais à esquerda da *pseudochave h(k)*
  4. Acessa o balde associado aos d bits obtidos
  5. Se há espaço no balde
     - 5.1. Insere a pseudochave no balde
  6. Senão
     - 6.1. Se $d_{local} = d_{global}$, duplica o tamanho do diretório
     - 6.2. Realiza a divisão do balde e incrementa d_{local}

Hashing extensível

- Casos da inserção
  - Há espaço no balde (caso mais simples)

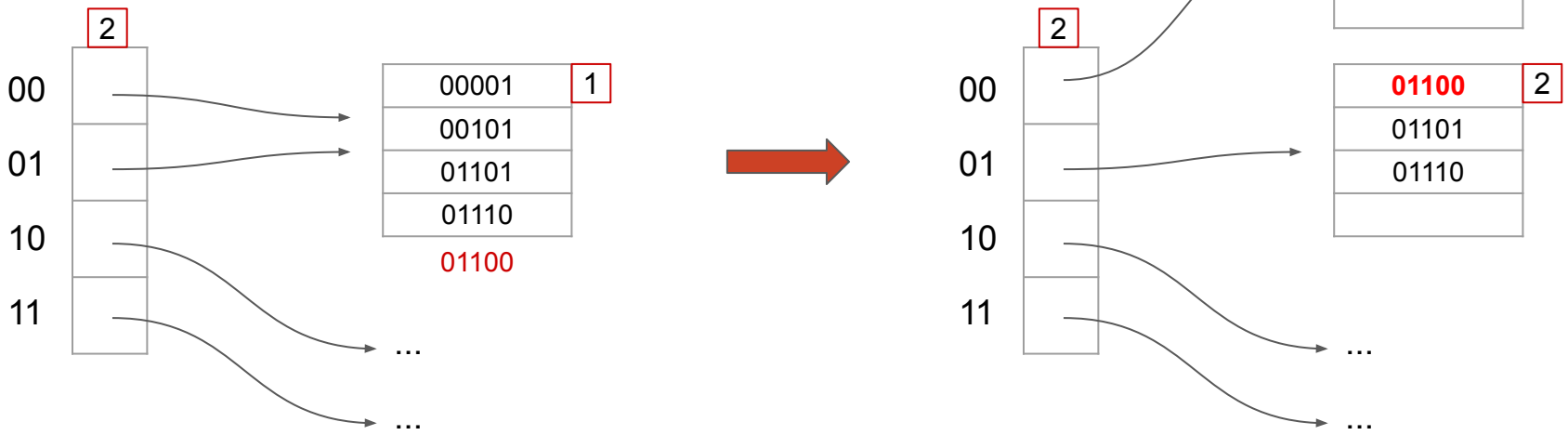
Inserir a chave: 01110



Hashing extensível

- Casos da inserção
  - Balde cheio e $d_{local} < d_{global}$

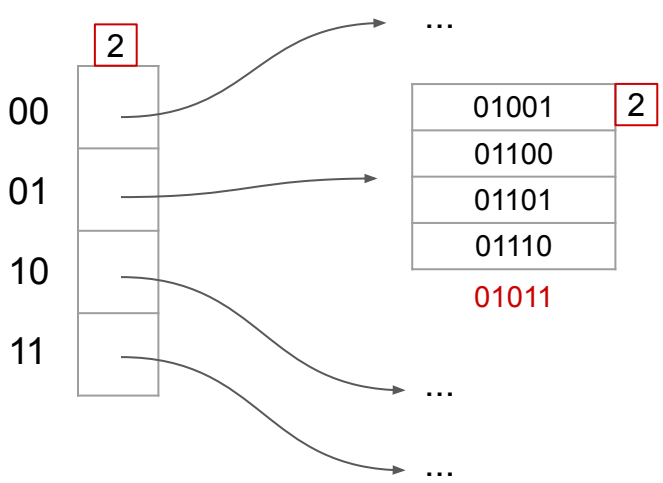
Inserir a chave: 01100

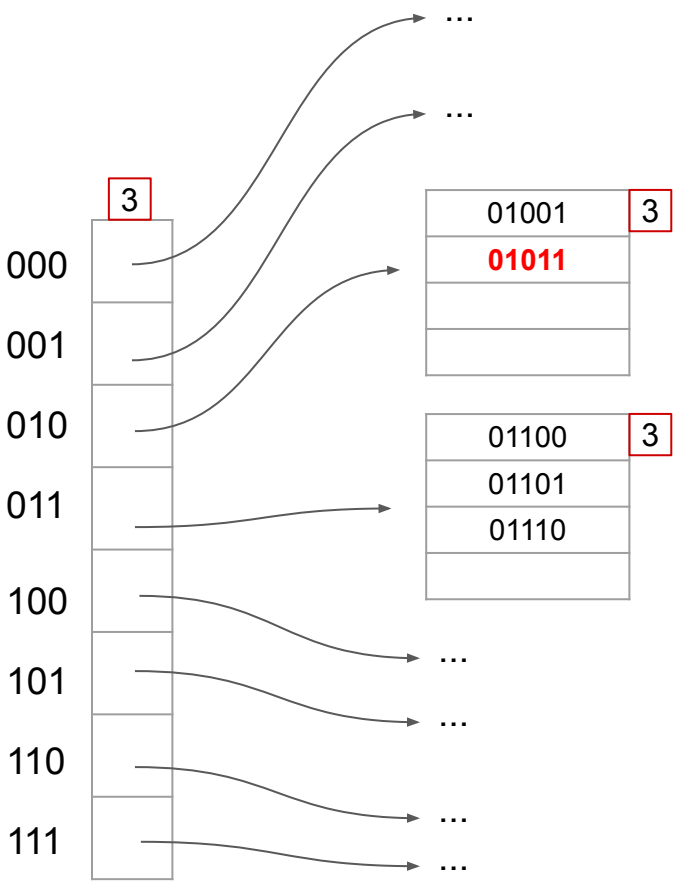


Hashing extensível

- Casos da inserção
  - Balde cheio e $d_{local} = d_{global}$

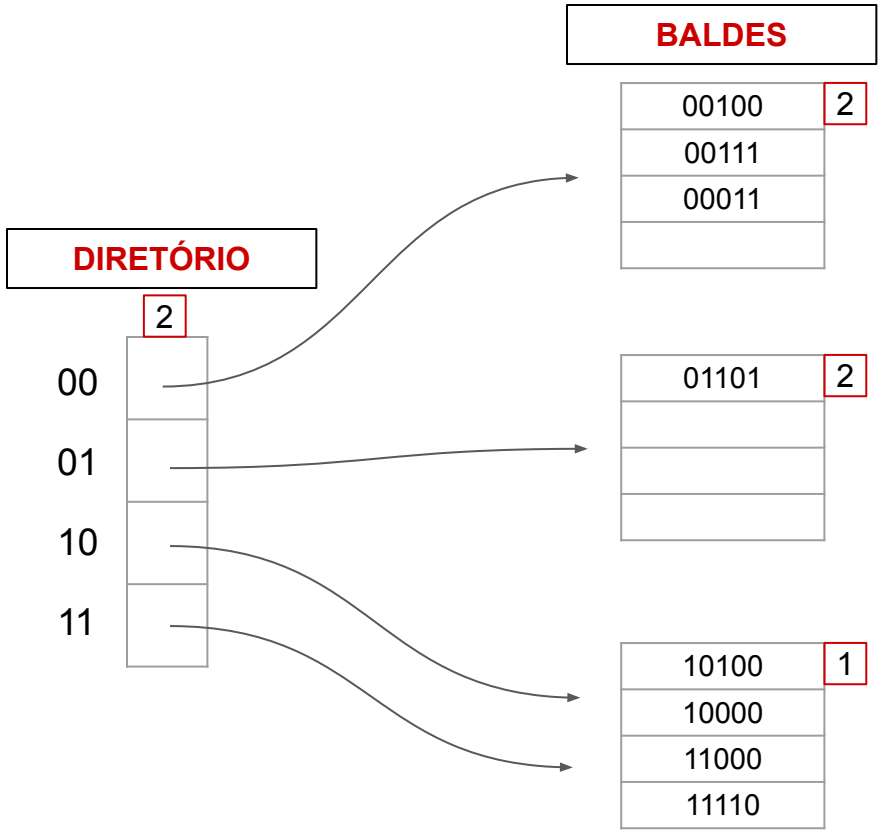
Inserir a chave: 01011





Hashing extensível

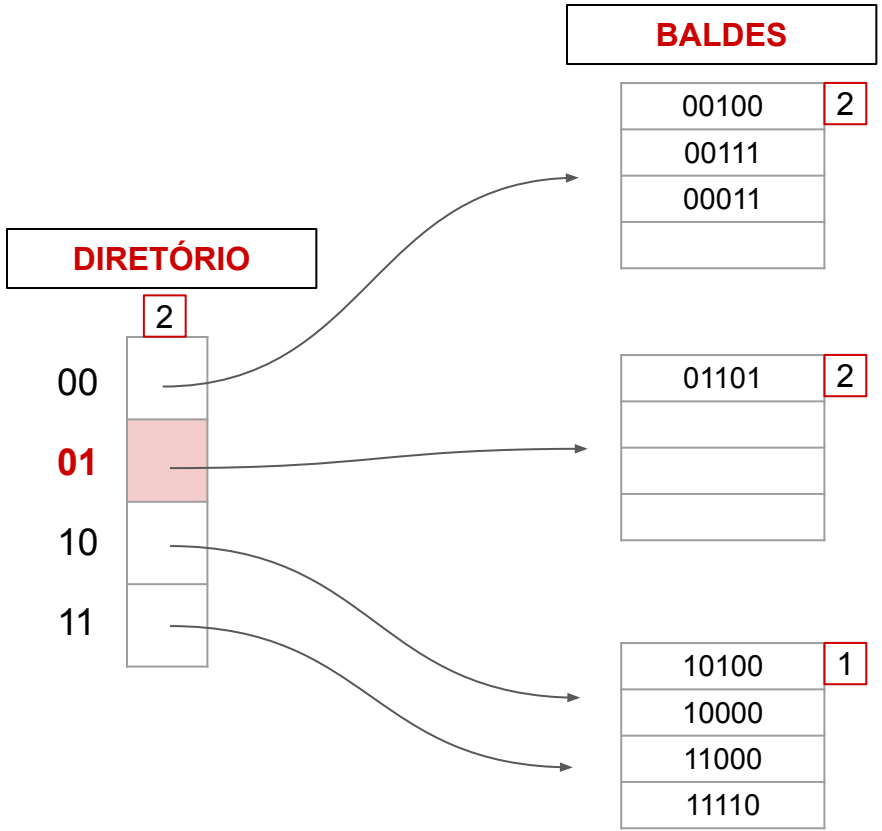
➡ **Inserir 01100**



Hashing extensível

Inserir 01100

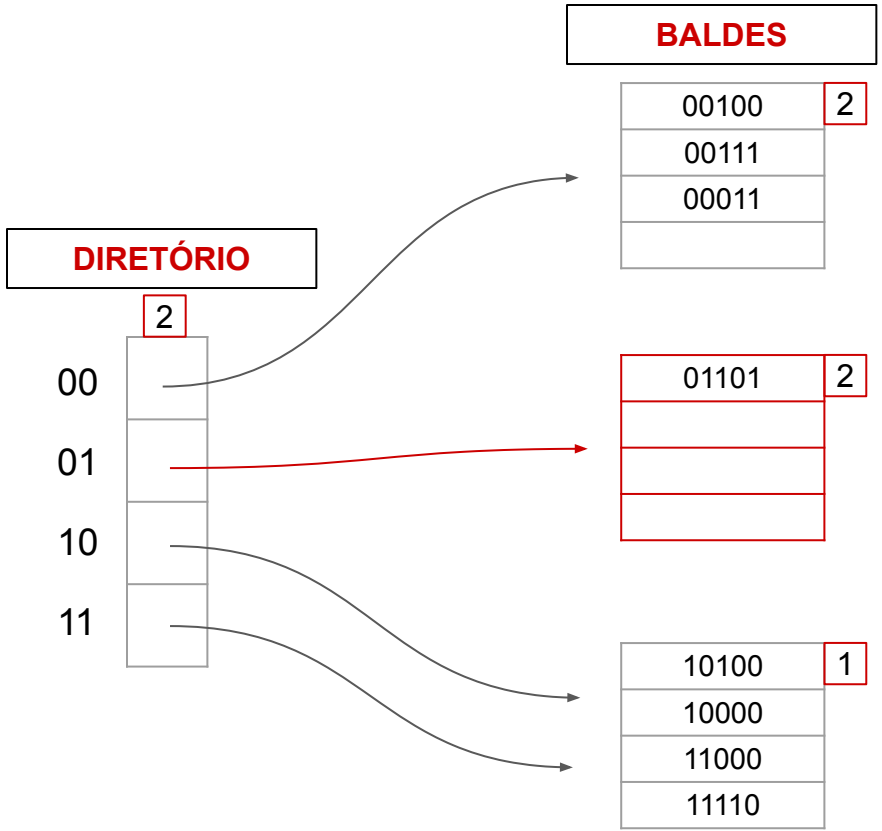
```
1.  Identifica os d bits mais à
    esquerda da pseudochave
```



Hashing extensível

Inserir 01100

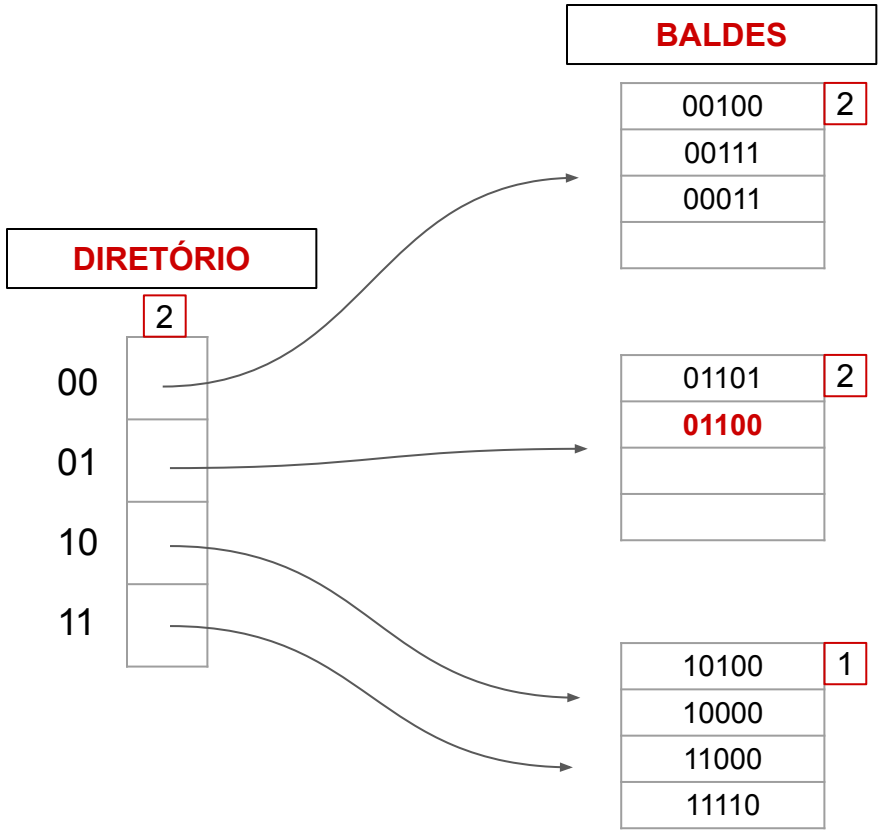
1. Identifica os d bits mais à esquerda da pseudochave
2. **Acessa o balde apontado por essa posição no diretório**



Hashing extensível

Inserir 01100

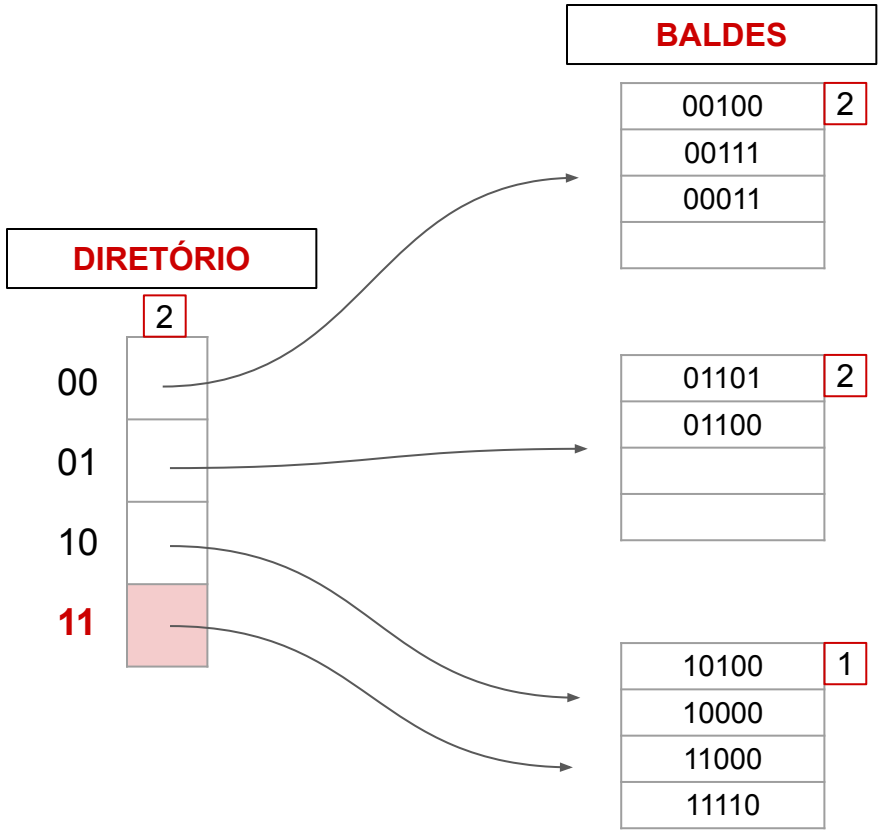
1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Balde tem espaço: insere**



Hashing extensível

Inserir 11111

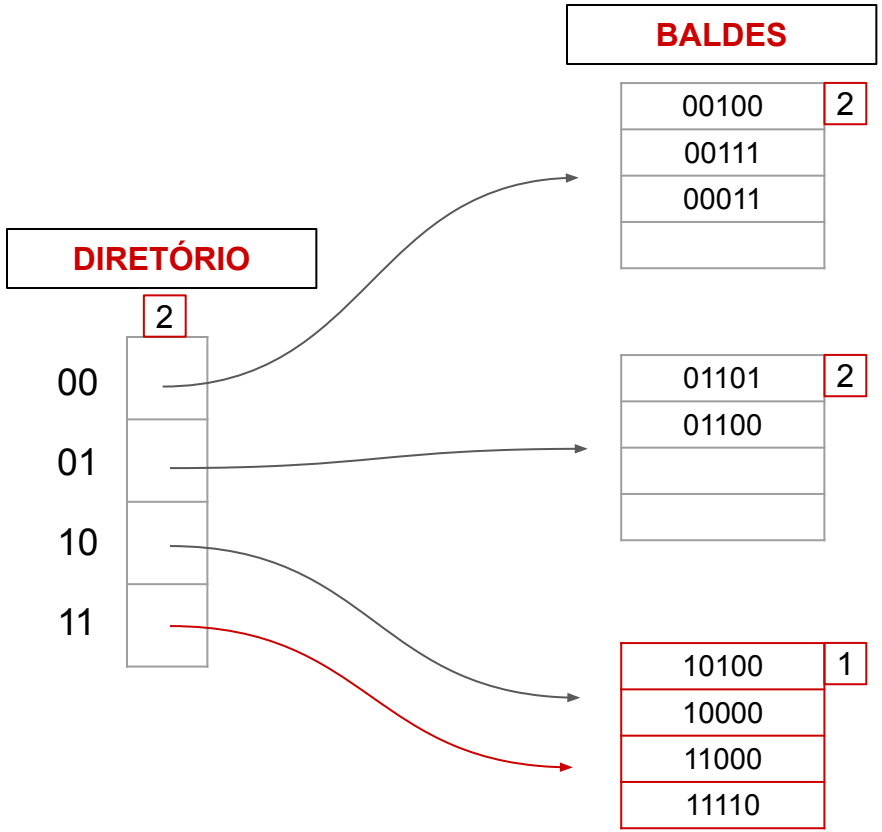
```
1.  Identifica os d bits mais à
    esquerda da pseudochave
```



Hashing extensível

Inserir 11111

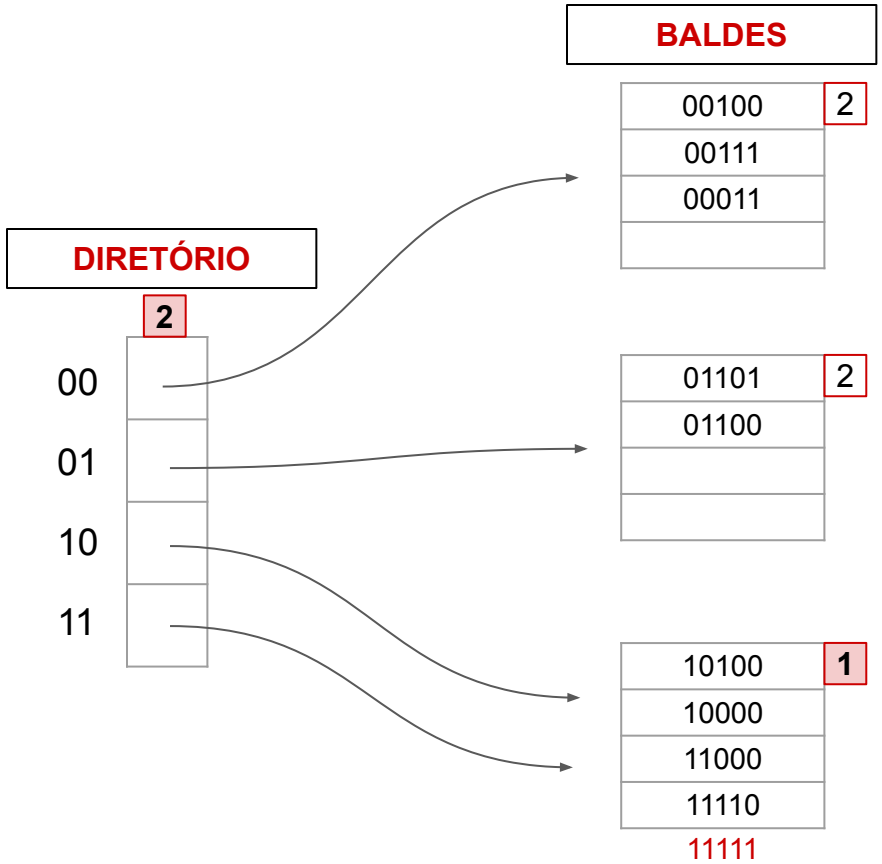
1. Identifica os d bits mais à esquerda da pseudochave
2. **Acessa o balde apontado por essa posição no diretório**



Hashing extensível

Inserir 11111

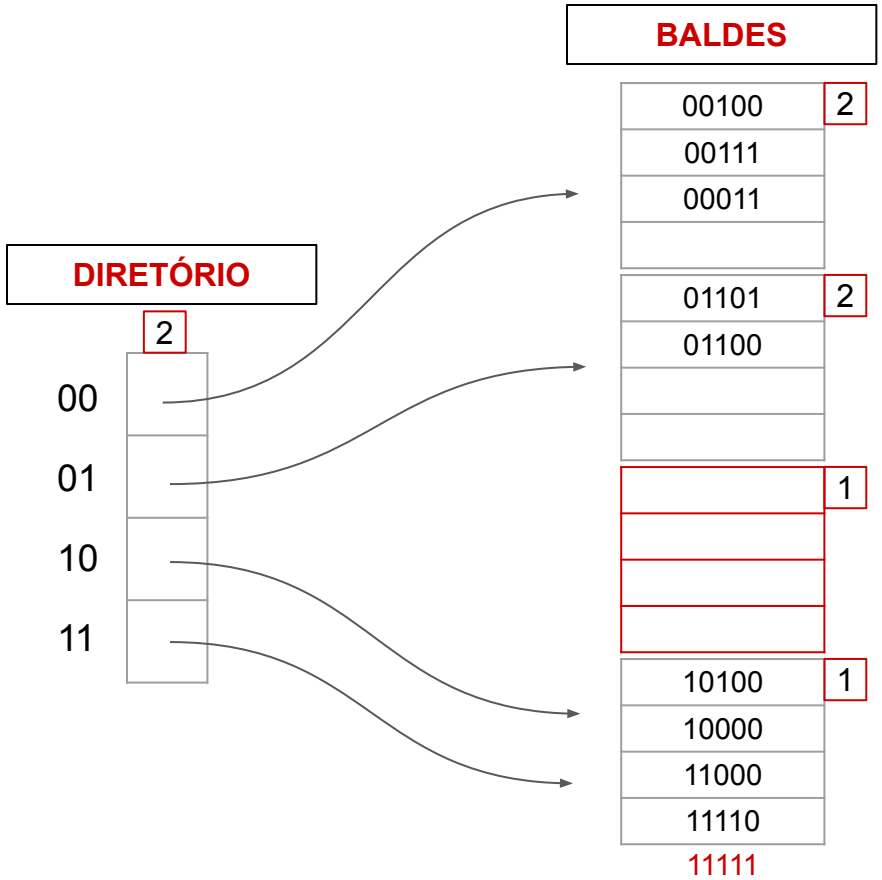
1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Balde cheio: $d_{local} < d_{global}$**



Hashing extensível

Inserir 11111

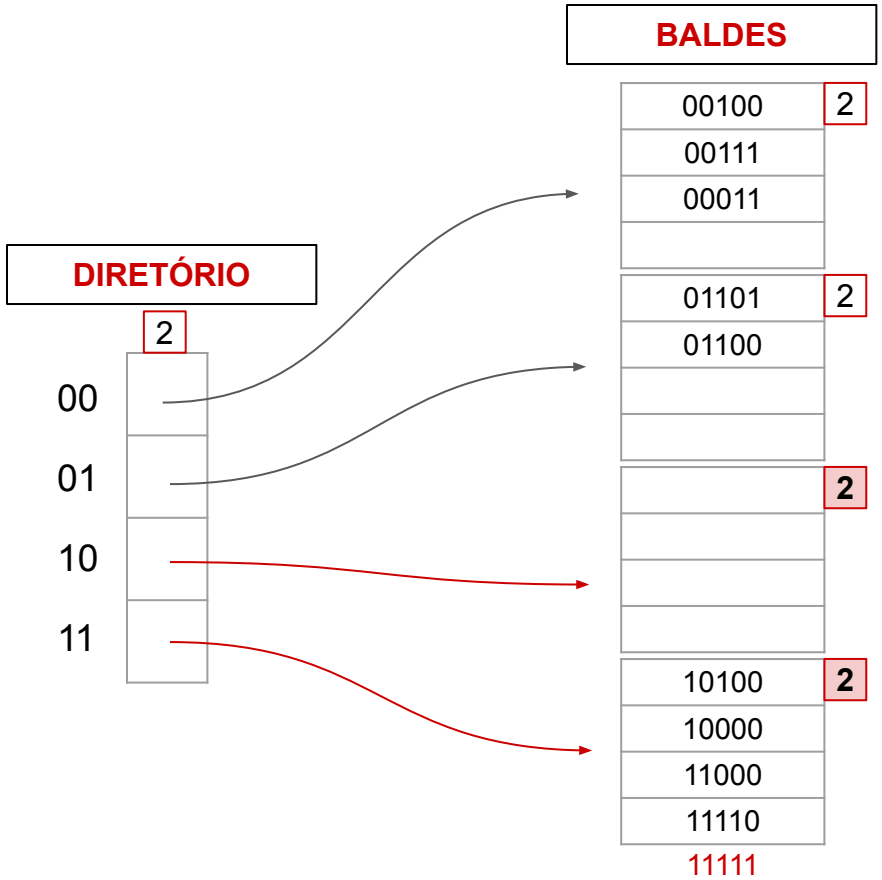
1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Balde cheio: $d_{local} < d_{global}$**
   - 3.1. **Cria novo balde**



Hashing extensível

Inserir 11111

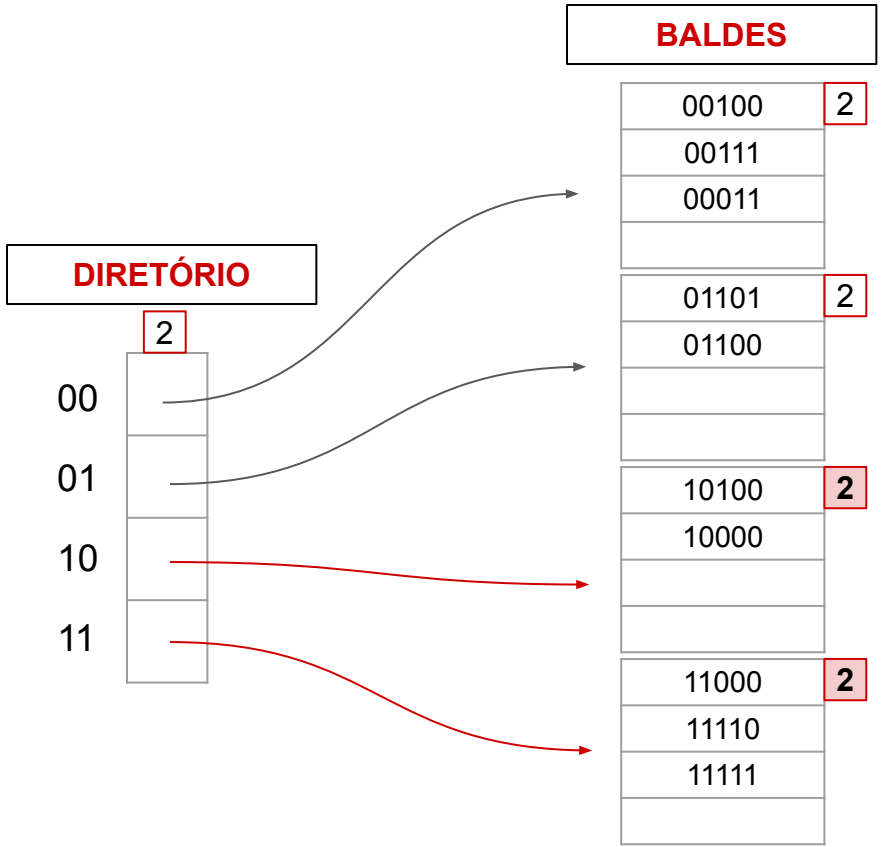
1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Balde cheio: $d_{local} < d_{global}$**
   - 3.1. Cria novo balde
   - **3.2. Ajusta ponteiros e d_{local}**



Hashing extensível

Inserir 11111

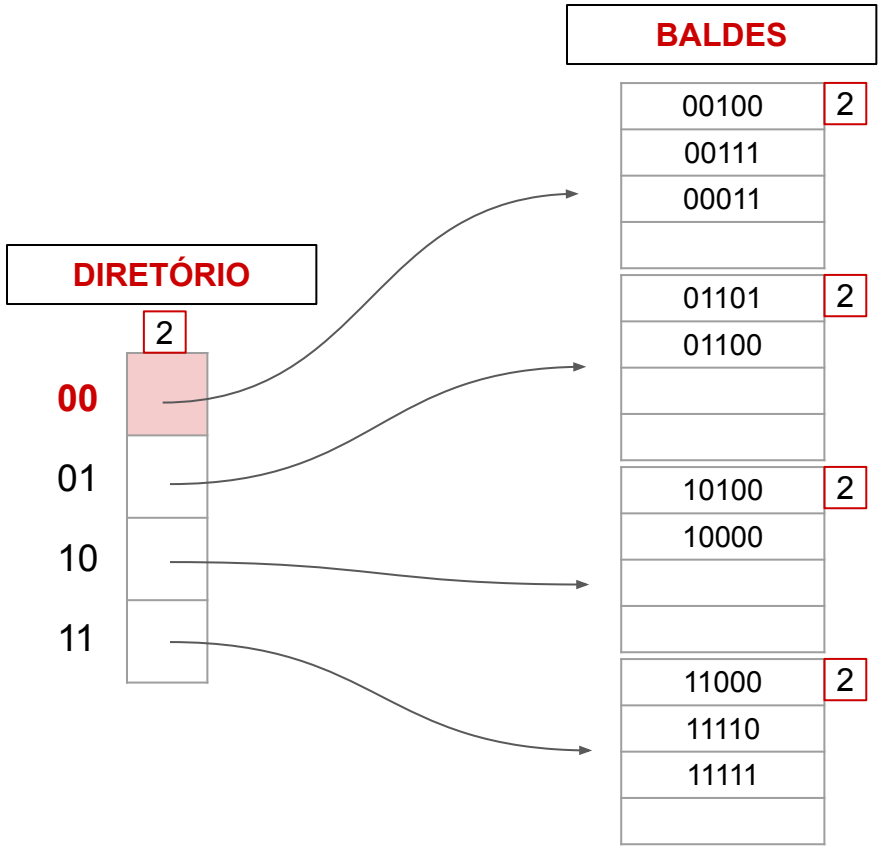
1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Balde cheio: $d_{local} < d_{global}$**
   - 3.1. Cria novo balde
   - 3.2. Ajusta ponteiros e d_{local}
   - **3.3. Redistribui as chaves**



Hashing extensível

Inserir 00000

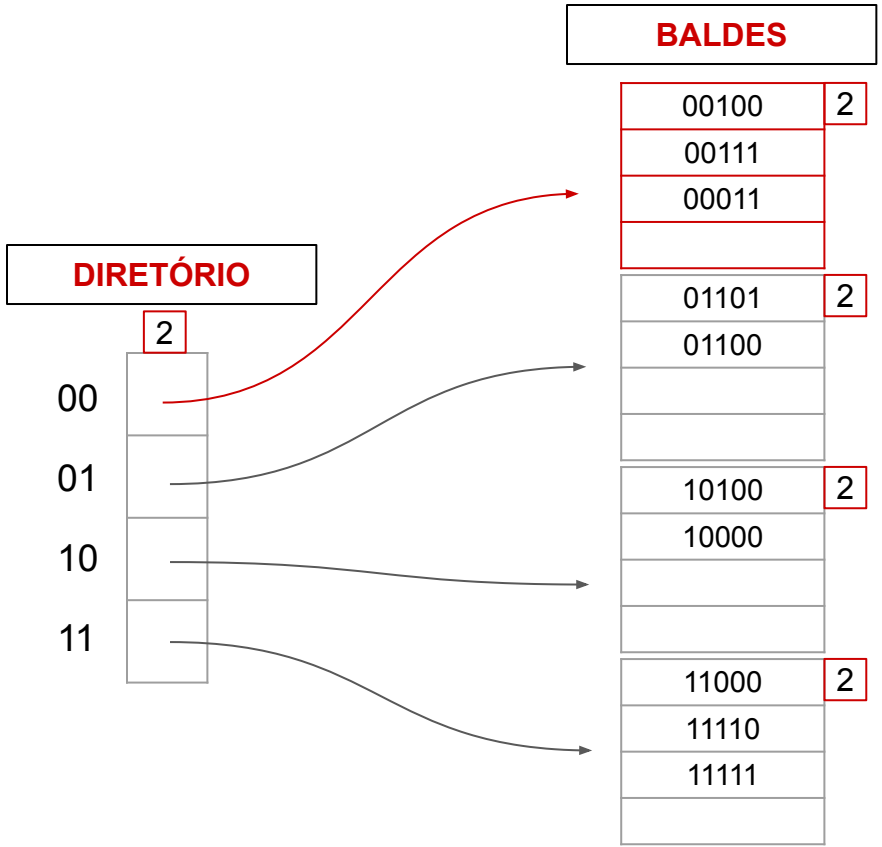
```
1.  Identifica os d bits mais à
    esquerda da pseudochave
```



Hashing extensível

Inserir 00000

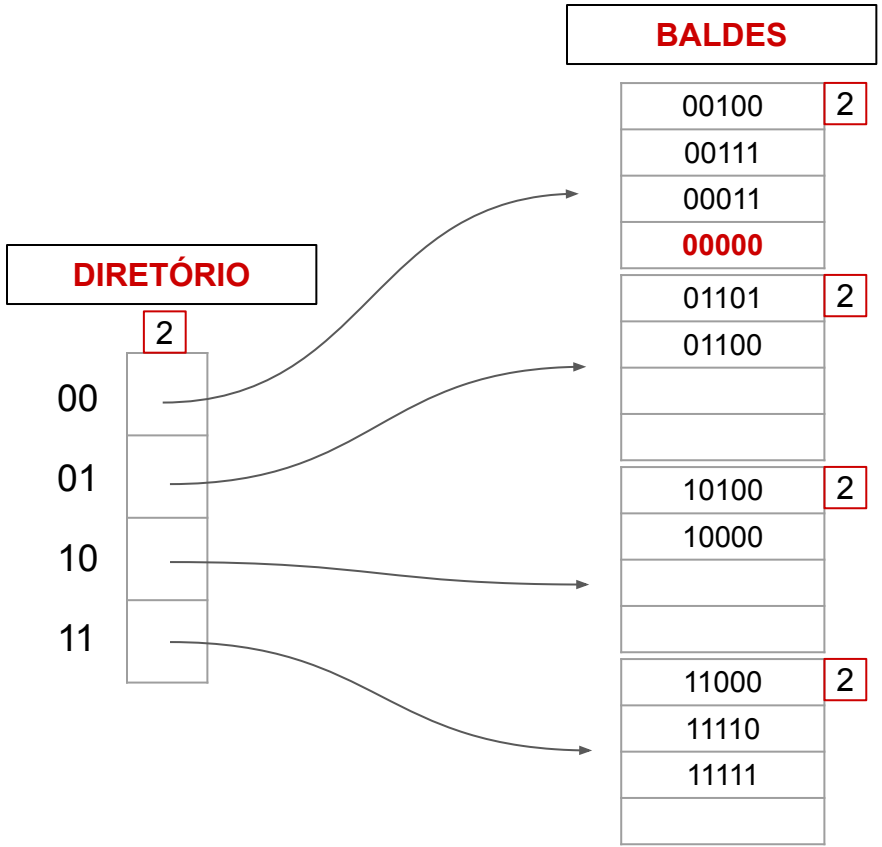
1. Identifica os d bits mais à esquerda da pseudochave
2. **Acessa o balde apontado por essa posição no diretório**



Hashing extensível

Inserir 00000

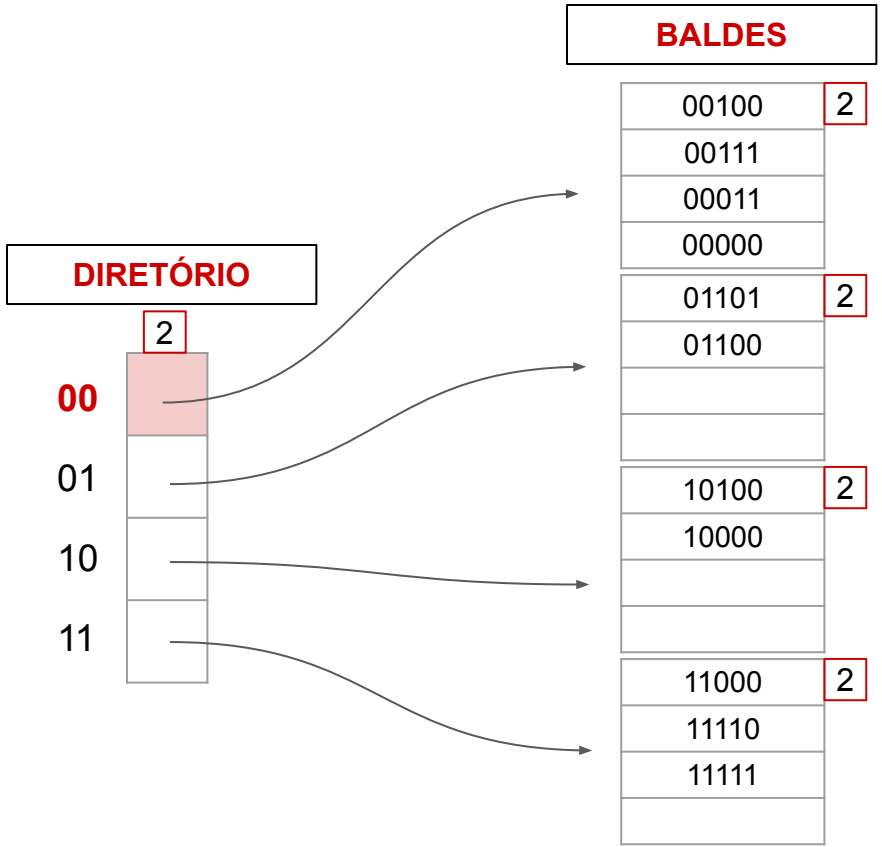
1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Balde tem espaço: insere**



Hashing extensível

Inserir 00001

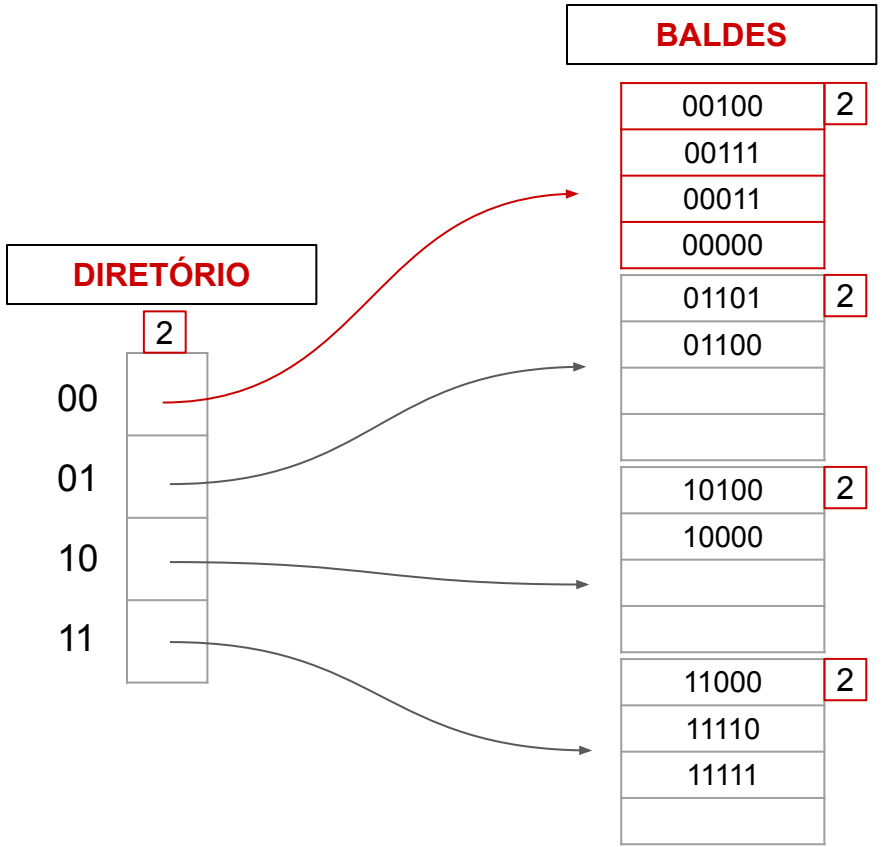
```
1.  Identifica os d bits mais à
    esquerda da pseudochave
```



Hashing extensível

Inserir 00001

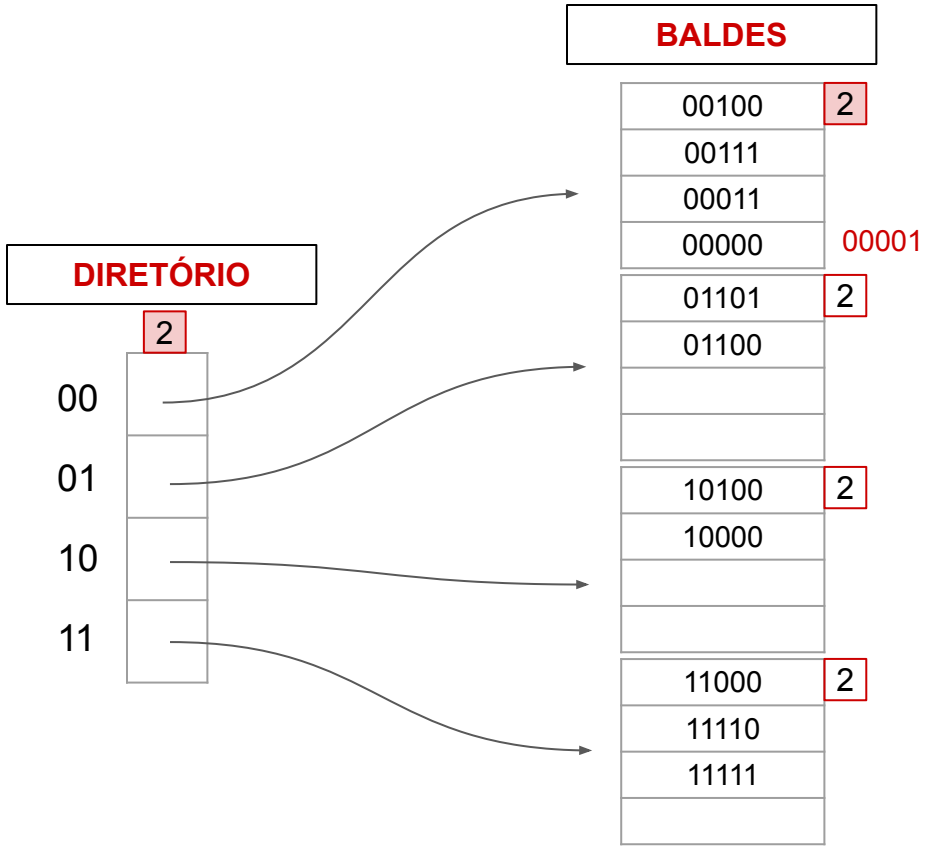
1. Identifica os d bits mais à esquerda da pseudochave
2. **Acessa o balde apontado por essa posição no diretório**



Hashing extensível

Inserir 00001

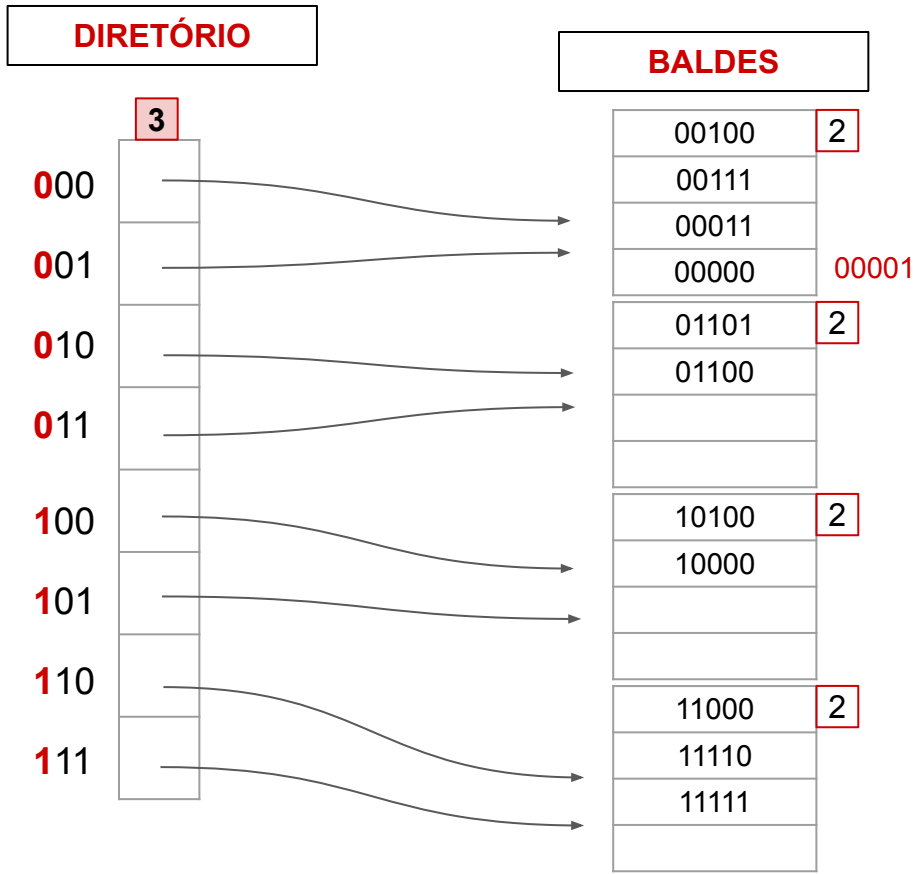
1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Balde cheio: $d_{local} = d_{global}$**



Hashing extensível

Inserir 00001

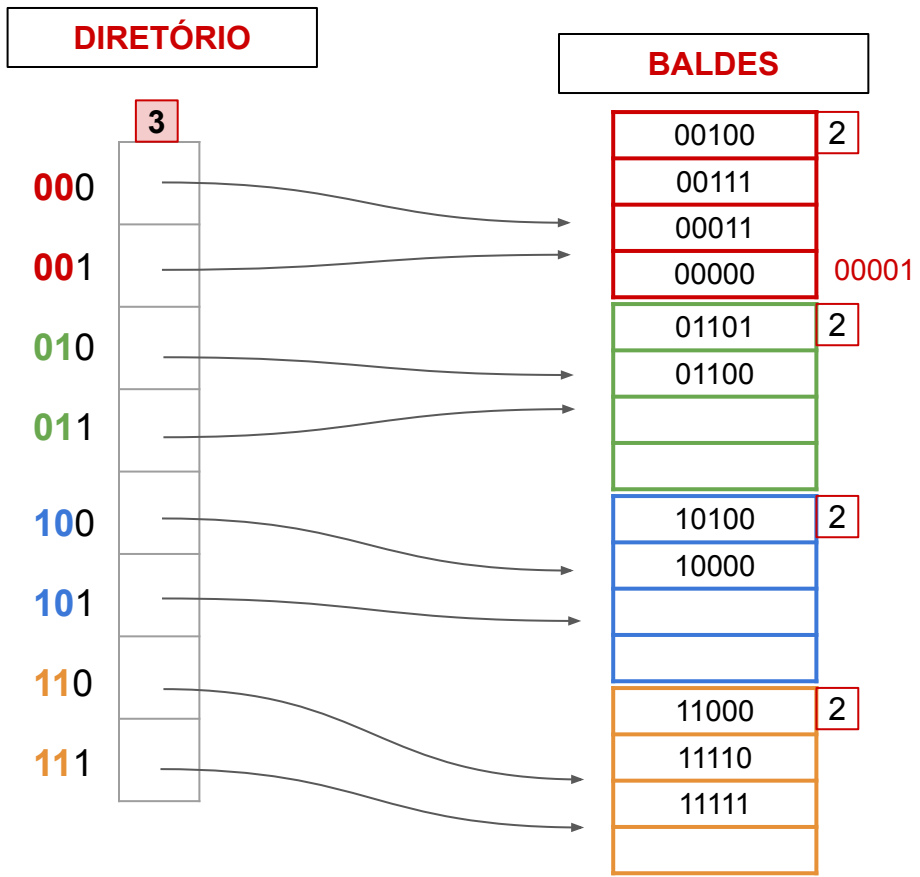
1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Balde cheio: $d_{local} = d_{global}$**
   - 3.1. **Duplica diretório**



Hashing extensível

Inserir 00001

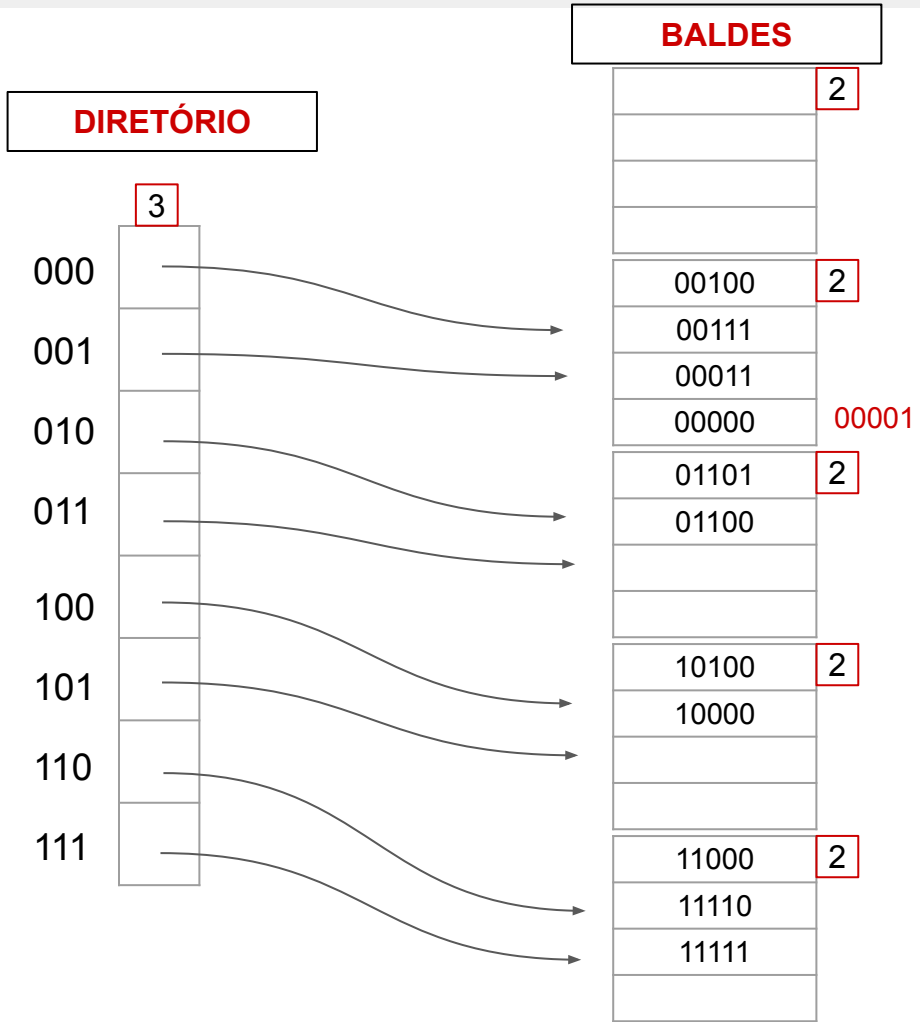
1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Balde cheio: $d_{local} = d_{global}$**
   - 3.1. **Duplica diretório**



Hashing extensível

Inserir 00001

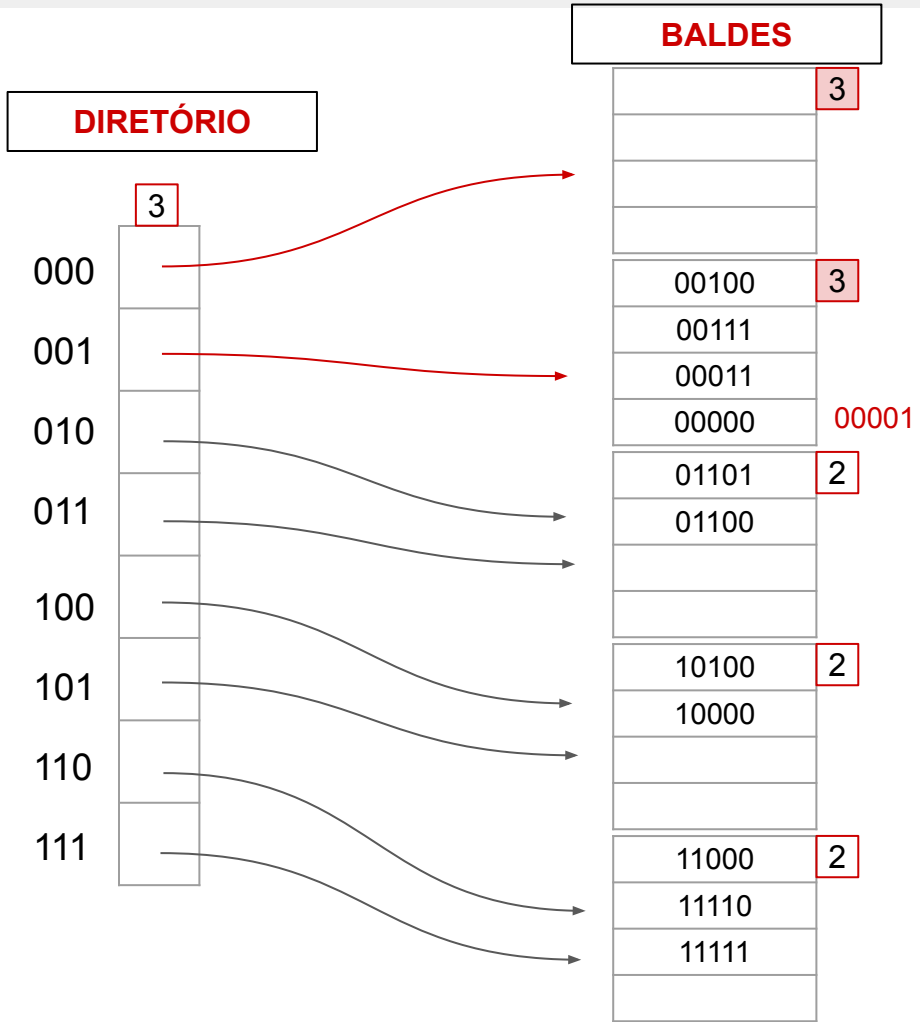
1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Balde cheio: $d_{local} = d_{global}$**
   - 3.1. Duplica diretório
   - **3.2. Cria novo balde**



Hashing extensível

Inserir 00001

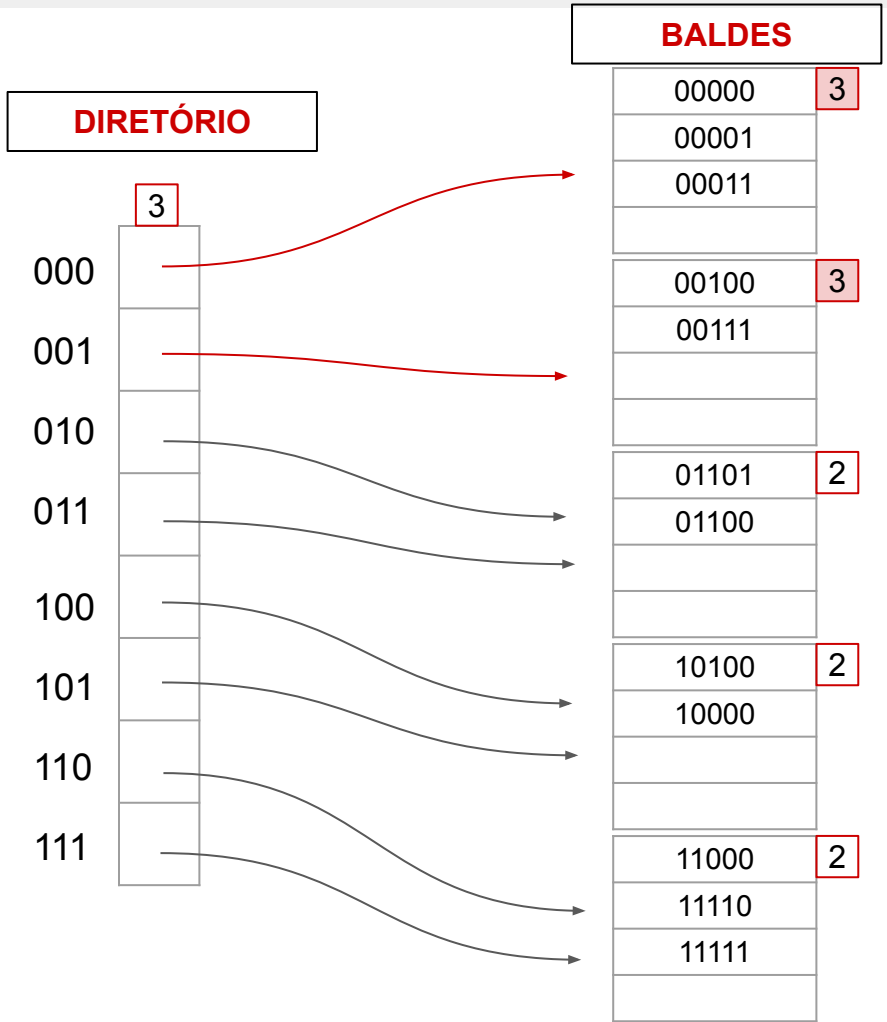
1. Identifica os d bits mais à esquerda da *pseudochave*
2. Acessa o balde apontado por essa posição no diretório
3. **Balde cheio: $d_{local} = d_{global}$**
   - 3.1. Duplica diretório
   - 3.2. Cria novo balde
   - **3.3. Ajusta ponteiros e d_{local}**



Hashing extensível

Inserir 00001

1. Identifica os d bits mais à esquerda da pseudochave
2. Acessa o balde apontado por essa posição no diretório
3. **Balde cheio: $d_{local} = d_{global}$**
   - 3.1. Duplica diretório
   - 3.2. Cria novo balde
   - 3.3. Ajusta ponteiros e d_{local}
   - **3.4. Redistribui as chaves**



Hashing extensível

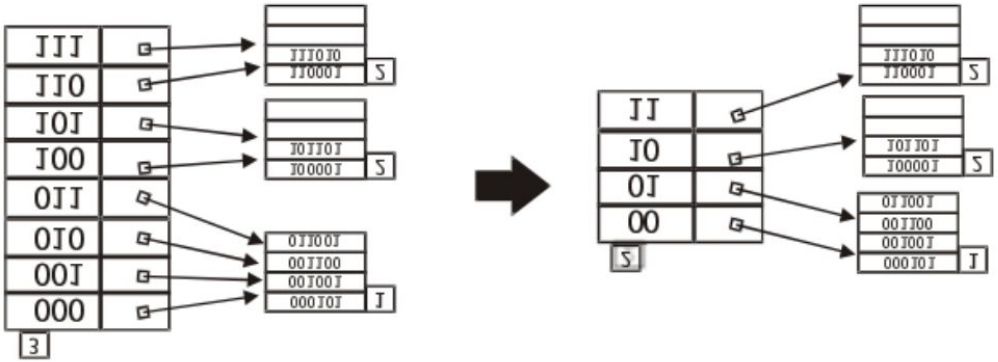
- Remoção
  - Possibilidade de fusão de baldes
    - Fundir sempre que possível?
    - Fundir somente se a quantidade de chaves for menor que algum valor?
  - A fusão é realizada com um **balde amigo**
    - Balde que difere apenas no último bit (mais à direita) do índice
    - $d_{local} = d_{global}$
  - Possibilidade de redução do diretório
    - Apenas quando **todos** os baldes possuem $d_{local} < d_{global}$

Hashing extensível - Remoção

- Ao remover uma entrada, temos que fazer duas verificações
  - Se é possível fundir o balde com um balde amigo
  - Se o tamanho do diretório pode ser reduzido
- Encontrando balde amigo
  - Para haver um único balde amigo, a profundidade do balde tem que ser a mesma do diretório.
  - Dado um índice para o balde, o balde amigo é aquele que difere apenas no último bit do índice
  - Exemplo: Se um balde tem índice 000 e sua profundidade é 3 e é igual a do diretório, o balde amigo tem índice 001.
  - Quando fundir?
    - Critério a cargo do programador
    - Pode-se sempre tentar fundir os baldes
    - Pode-se definir um número mínimo de informação em cada balde

Hashing extensível - Remoção

- Reduzindo o diretório
  - Depois de fundir o balde, talvez o diretório possa ser reduzido
  - É possível quando cada balde é referenciado por pelo menos duas entradas do diretório Isto é, toda profundidade local é menor que a profundidade global



Hashing extensível

- Vantagens do hashing extensível
  - Evita reorganização do arquivo quando ocorre overflow do diretório
    - Diretório é usualmente mantido na memória principal
- Problemas
  - Diretório pode crescer demais
  - Tamanho do diretório não cresce uniformemente

Exercício

- Insira a seguinte sequência de *pseudochaves*, na ordem especificada:
  - 00001, 01100, 10000, 10001, 11001, 10101, 01111, 00100 e 11111
- Após as inserções, quais chaves devem ser removidas para forçar uma redução do diretório?
  - Mostre a remoção destas chaves
- Considere um diretório com profundidade inicial d = 1 e um balde com tamanho M = 2

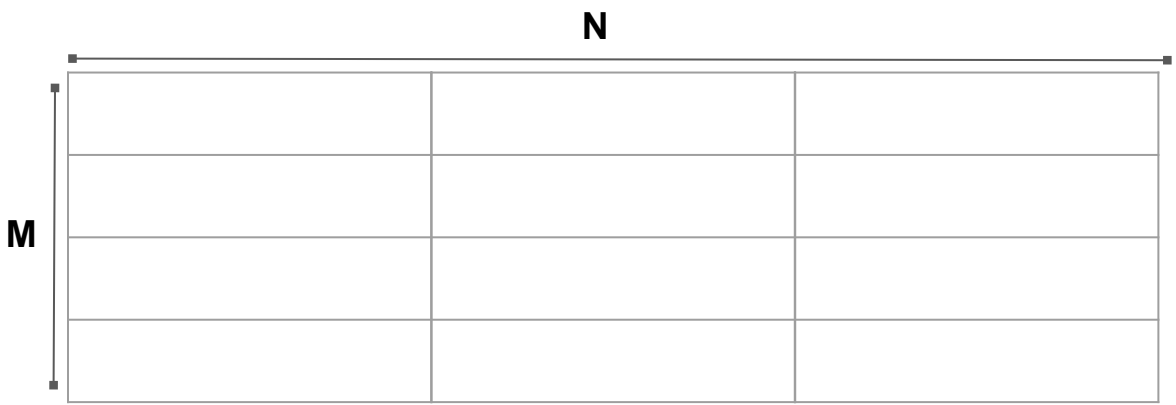
Hashing Linear

Introdução

- O hashing extensível cresce de forma exponencial no momento que cada balde precisa de mais registros
- Além disso, é necessário um dicionário para armazenar o endereço de cada balde
- O hashing linear cresce de forma mais lenta
  - É criado um balde a cada momento que a tabela cresce
  - Não é necessária uma tabela de endereçamento
  - Dessa forma, não é necessário também refazer todos os índices a cada crescimento
- Funcionamento
  - Manipulação de espaços através de diferentes funções de hash aplicadas nos espaços disponíveis

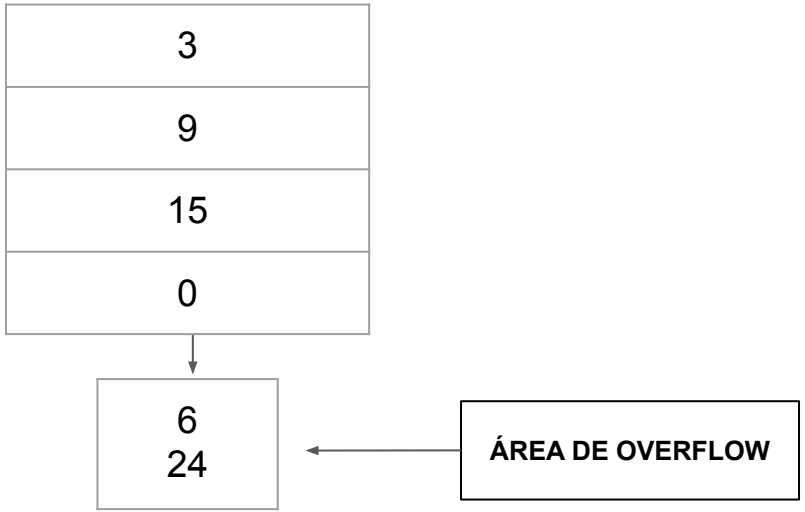
Introdução

- Essa abordagem também usa o conceito de baldes que são estruturas com espaços para mais de um registro
  - Equivale a “baldes alocados em uma estrutura sequencial”
  - Cada balde possui um tamanho fixo M, ou seja, armazena M chaves.
- Tabela hash inicia com um número fixo de N baldes.



Introdução

- Se a função de hash mapeia uma chave em um balde já cheio, pode-se usar uma **área de overflow**
  - Essa área pode ser uma lista encadeada



Hash Linear

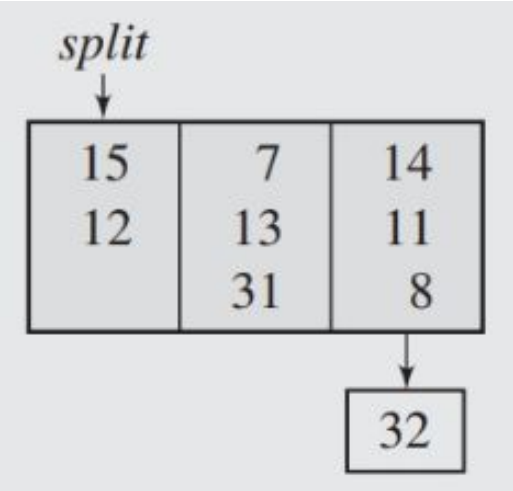
- A criação de novo baldes é determinado pelo fator de carga da estrutura
  - Ou seja, vários baldes podem conter mais registros do que o espaço determinado e fazer uso de área de overflow
  - A cada inserção, é verificado o fator de carga.
  - Caso seja maior do que um limite estipulado, um novo espaço de alocação é determinado.
- O fator de carga é dado pela quantidade de registros inseridos divididos pela soma dos espaços possíveis nos baldes:
  - FatorCarga = Total(registros) /(N*M + O_{size})

Hash linear - Inserção

- A inserção é dada pela função hash
  - $h_g(k) = k \bmod 2^g * N$, sendo g iniciado de 0.
- Ao inserir uma nova chave, aloca-a no balde determinado pela fórmula acima
  - Caso o balde esteja cheio, a chave é alocada no área de *overflow*
  - Quando o fator de carga ultrapassa o máximo valor estabelecido, duplica-se o balde apontado por split.
  - O novo balde é alocado no final da tabela e o split é incrementado.
  - Os baldes duplicados são reorganizados e todos os dados dos dois baldes passam a obedecer a fórmula $h_{g+1}(k) = k \bmod 2^{g+1} * N$.

Inserção - Exemplo

Suponha M=3, N=3 e fator de carga = 80% e área de overflow de tamanho 1.
Suponha a tabela abaixo já preenchida com algumas chaves e ponteiro split apontando para o índice 0 da tabela.



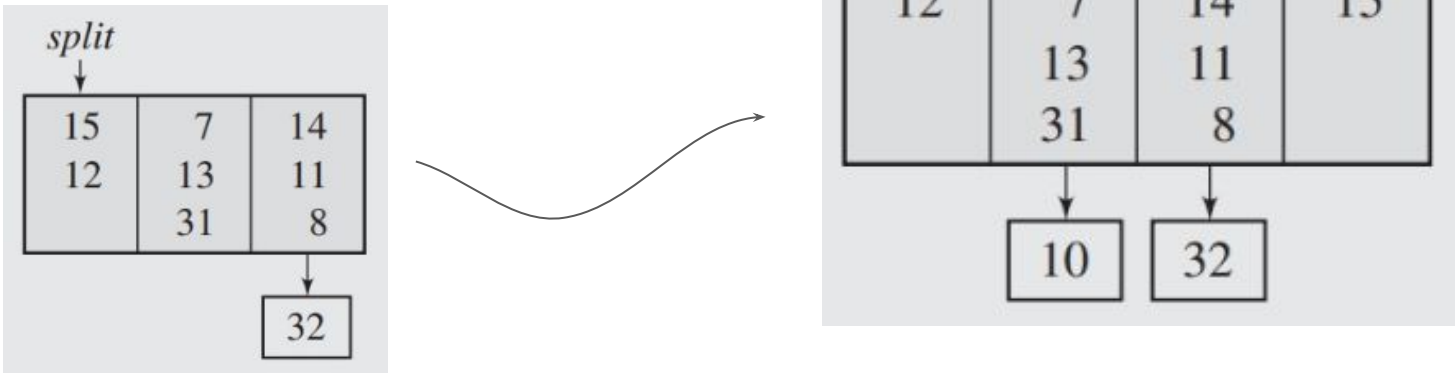
Fator de carga da tabela é 75% = 9 chaves / (9 baldes + 3 área overflow) = 9/12.

Como a tabela ainda não sofreu expansão (nível g = 0, N=3 é o valor inicial), todas as chaves foram alocadas com a função:

- $h_0(k) = k \bmod 2^0 * 3$
- $h_0(k) = k \bmod 3$

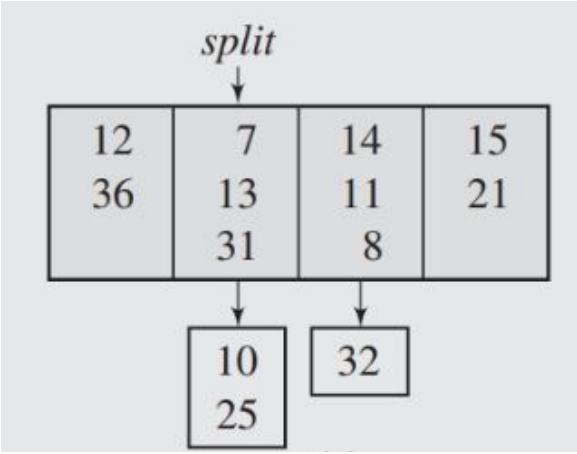
Inserção:

- Inserindo 10, o fator de carga vai para 83% (10/12)
  - Torna-se necessário criar um novo balde
  - O novo bucket é alocado no final da tabela. As chaves dos dois buckets são redistribuídas com a fórmula:
    - $h_1(k) = k \bmod 2^{0+1} * 3$.
    - $h_1(k) = k \bmod 6$
  - O ponteiro split é incrementado.

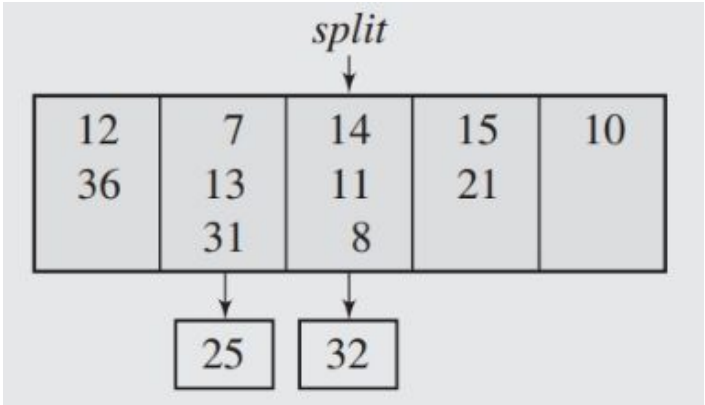


Inserção

- Inserindo 21, 36 e 25.
  - O fator de carga vai para 87%

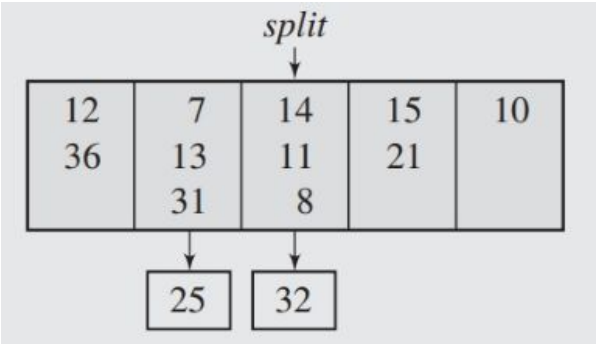


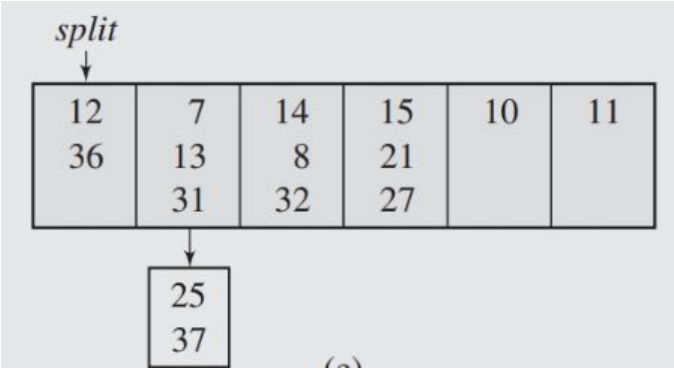
1. O novo bucket é alocado no final da tabela.
2. As chaves dos dois baldes são redistribuídas com a fórmula
   - $h_1(k) = k \bmod 2^{0+1} * 3$.
   - $h_1(k) = k \bmod 6$
3. O ponteiro p (split) é incrementado



Inserção

- Inserindo 27 e 37.
  - O fator de carga vai para 83%.
  - As chaves dos dois baldes são redistribuídas com a fórmula
    - $h_1(k) = k \bmod 2^{0+1} * 3$.
    - $h_1(k) = k \bmod 6$
  - Como split é igual a $N*2^g$, o ponteiro split volta para a posição inicial e incrementa g.





Inserção

1. Calcula h = $h_g(k)$
2. Se h < split, calcula h = $h_{g+1}(k)$
3. Se há espaço no balde h
   3.1. Insere k no balde
4. Senão
   4.1. Insere k na área de overflow
5. Se fator de carga máximo excedido
   5.1. Cria um novo balde
   5.2. Redistribui as chaves do balde indicado por split usando $h_{g+1}(k)$
   5.3. Incrementa split
   5.4. Se split é igual a N ⇒ volta com split pra 0 e incrementa g

Considerações

- Hashing linear necessita de uma área de overflow
  - A ordem de divisão é pré-determinada
  - A área também pode ser implementada de forma coalescida
- Assim como no hashing extensível, a tabela cresce ao dividir baldes, mas não há necessidade de um diretório
  - Mais rápido e consome menos espaço
    - Não há necessidade de recomputar os índices
    - Hashing extensível cresce de forma exponencial
  - Eficiência em arquivos grandes

Exercício

- Insira a seguinte sequência de chaves, na ordem especificada:
  - 2, 21, 14, 11, 6, 9, 30, 22, 7, 15, 0, 1, 44, 26, 17, 19 e 31
  - Considere uma tabela com N = 2 e M = 3. Considere um fator de carga de 75% como limite, porém sem contabilizar os espaços da área de overflow no cálculo
- Refaça o exercício, agora considerando que o critério para divisão não é mais o fator de carga, e sim a presença de algum balde cheio

Referências

- DROZDEK, Adam. Data Structures and Algorithms in C++, Fourth Edition, cap. 10. Cengage Learning, 2013.
- SOUZA, Jairo F. Notas de aula de Estrutura de Dados II. 2016. Disponível em: http://www.ufjf.br/jairo_souza/ensino/material/ed2/